UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL
Y DE POSGRADO
DEL COLEGIO DE CIENCIAS Y HUMANIDADES
INSTITUTO DE INVESTIGACIONES EN MATEMATICAS
APLICADAS Y SISTEMAS

INTERFAZ DE COMUNICACION PARA UNA RED COOPERATIVA

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS
DE LA COMPUTACION

P R E S E N T A :

HECTOR ALEJANDRO DURAN LIMON

DIRECTOR: DR. VICTOR GERMAN SANCHEZ

MEXICO D. F. FEBRERO 1996

TESIS

COMPLETA

Agradecimientos

- A mis padres, Diego Durán Ordaz y Aurora Limón Velazquez, asi como a mis hermanos, por haberme brindado todo su apoyo y su amor.

- Al Dr. Victor Germán Sánchez, por haber dirigido esta tesis y por todo el apoyo que me brindó. Al Dr. Christian Lemaitre, también por su apoyo y asesoría. A la Dra. Cristina Loyo por el apoyo que me proporcionó cuando realicé mi tesis en el LANIA.
  Al Mtro. Horacio Carvajal y al Mtro. Refugio Vallejo por haber aceptado ser mis sinodales y además por sus comentarios en el trabajo de tesis.

- A la Dra. Hanna Oktaba, por su empeño en mantener en un buen nivel a la Maestría.

- A Edna Palma Cossaín, por todo el apoyo y su amor que me ha proporcionado y por haber realizado una revisión del manuscrito del trabajo.

- A mis jefes, el Ing. Victor Jiménez y la Ing. Gabriela Medina, por haberme apoyado con la licencia que se me otorgó para poder realizar mi tesis en el LANIA.

- A todos mis maestros y amigos de la Maestría.

Contenido

Capítulo 4
Diseño de la IC y del SAP

Capítulo 5
Especificación de la IC

Introducción

El trabajo cooperativo puede conducir a la formación de grupos de agentes, los cuales interaccionan unos con otros a través de la comunicación. Y mediante ésta, y a partir de sus necesidades particulares, llegan a la formulación de un objetivo común. De donde, opcionalmente, se fabrica un plan, él cual finalmente designará la repartición de tareas entre los agentes.

Los sistemas cooperativos resultan útiles en aquellos problemas que son inherentemente distribuidos, es decir, que su distribución es natural; como por ejemplo un sistema de radar que cubra varias regiones de un plano.
También son adecuadas para aumentar la velocidad y eficiencia de algunos sistemas.
La distribución del control de un sistema, permite aumentar su confiabilidad.
Hay problemas que al crecer sus dimensiones se ven limitados en los recursos que poseen, sin embargo, con la distribución de recursos pueden ser superados.

Los problemas básicos que enfrenta un sistema cooperativo son: la formulación, descripción, descomposición, y asignación de tareas. Las dos primeras permiten definir el problema a resolver, así como sus límites.
Mediante los dos últimos se lleva a cabo la descomposición de una tarea global en subtareas para asignarlas a los agentes.
Existen otro tipo de problemas como son: el control de la coordinación, mantener la coherencia, el proceso de comunicación entre agentes, etc.

Se ha desarrollado una serie de mecanismos para lograr la solución cooperativa de problemas. Entre estos los más importantes son: la Red de Contratos, el esquema de Pizarrón, Organizaciones, y Modelos.
Cada uno de ellos se adecúa mejor a un tipo de problema.
El primero lleva a cabo la asignación de tareas a través del anuncio de éstas, entablando posteriormente una negociación, y finalizando con un

contrato.
En el segundo se tiene una área de trabajo llamada pizarrón, la cual está dividida en varios niveles, donde cada uno de estos corresponde a una parte del dominio del problema. En éstos, los agentes pueden leer o escribir, y son llamados fuentes de conocimiento.
A través de la escritura y lectura de hipótesis, eventualmente se llega a una solución.

Por otro lado las organizaciones plantean introducir el concepto de organización como fundamental en la estructura de un sistema cooperativo.
Y finalmente los Modelos pretenden modelar un comportamiento distribuido como una sociedad de agentes.

Un sistema cooperativo requiere de un sistema distribuido (SD) como plataforma y apoyo a la interacción que llevan a cabo los agentes que la conforman.
Dicha plataforma permite el intercambio de mensajes entre los agentes que integran un grupo de trabajo cooperativo.

Los tipos de comunicación que pueden existir en un sistema distribuido son: paso síncrono y asíncrono de mensajes, comunicación generativa, las RPC's y Rendezvous.
En la primera, el transmisor espera hasta que el receptor haya leído el mensaje, a diferencia de la segunda, en donde el transmisor no realiza este tipo de espera.
La comunicación generativa pretende que el trasmisor envíe todos los mensajes a una entidad mediadora, en donde los receptores pueden ir a buscarlos. Los mensajes no van dirigidos a un agente en particular, sino al que cumpla con un determinado patrón.
Las RPC's y Rendezvous ofrecen realizar llamados a procedimientos remotos, como si estos fueran locales.

Existe una gran cantidad de modelos o lenguajes distribuidos, entre estos tenemos a: CSP, Actores, Linda, y Glish.
El primero permite la comunicación síncrona y unidireccional. Sin embargo, éste no permite la creación dinámica de procesos, ya que se encuentra fijada al inicio del programa.

Actores es un lenguaje basado en objetos, en donde todo lo que existe en este modelo es un actor.
La comunicación que se da entre estos es a través del paso de mensajes.

Linda sigue el modelo de comunicación generativa. En este lenguaje, un mensaje es representado por un tuplo, es decir, un conjunto de argumentos actuales y formales; los cuales determinan un patrón.
Los agentes transmisores mandan todos sus mensajes a una entidad llamada "espacio de tuplos". A ésta, los agentes receptores, le comunican el patrón que deben cumplir aquellos mensajes que desean recibir.

En Glish también existe una entidad que recibe todos los mensajes enviados por los agentes transmisores. Sin embargo, aquí sí llevan indicado el destinatario. Entonces la función de dicha entidad consiste en realizar algún tipo de procesamiento a todo mensaje que sea enviado.

Por todo lo anterior, comprobamos que un sistema cooperativo requiere de uno distribuido para que pueda funcionar adecuadamente.
Dicho sistema distribuido lo podemos ver como una red de agentes cooperativos (RAC), donde cada uno de los agentes posee las habilidades necesarias para comunicarse con los demás y permita administrar la coherencia en el proceso cooperativo.

Para ésto, el Laboratorio Nacional de Informática Avanzada, A.C. (LANIA), ha desarrollado una arquitectura para dichos agentes, llamada *facilitador*.
El modelo del *facilitador* está basado en la matáfora de oficina, la cual plantea la existencia de un *jefe* y una *secretaria*. El primero representa al *usuario* y el segundo al *facilitador*. Donde un *usuario* puede ser un proceso informático o un humano.

De manera que la *secretaria* le ayuda al *jefe* a organizar el trabajo cooperativo con otros *jefes*, entablando diálogos con otras *secretarias*.

La arquitectura del *facilitador* consiste de los siguientes módulos:

- la máquina de acciones (MA)
- la máquina de planes (MP)

- el servidor de acciones privadas (SAP)
- la interfaz del *usuario* (IU)
- la interfaz de comunicaciones (IC)

El primero se encarga de conmutar acciones (comunicativas y privadas) entre el *usuario*, la IC, el SAP y la MP.
El segundo es el ejecutor de tareas cooperativas de apoyo al *usuario*.
El SAP proporciona diferentes servicios informáticos locales y distribuidos.
La IU permite la interacción entre el *usuario* y la MA.
La IC ofrece los servicios básicos que permiten establecer comunicación con otros *facilitadores* de la red.

El objetivo general, de la presente tesis, es definir un ambiente de sistema distribuido para el apoyo de los sistemas cooperativos, a partir de lo siguiente:

1. La utilización de la arquitectura del *facilitador*.
2. La integración de los servicios distribuidos existentes.
3. El manejo de grupos dinámicos de trabajo, los cuales consideramos son de gran ayuda para el trabajo cooperativo.
4. La utilización de un servicio que permita la localización de un usuario en la red de comunicaciones de un sistema distribuido.

Para lograr lo anterior se realizará el diseño, especificación e implantación de la IC. Y el diseño y una pequeña implementación del SAP.
En el capítulo 1 se presenta un análisis y un panorama general de lo que son los sistemas cooperativos, así como sus principales mecanismos de operación. Ésta sección tiene como finalidad mostrar las características esenciales y la problemática que presentan los sistemas cooperativos.

En el capítulo 2 se ilustra un esquema general de los sistemas distribuidos, presentandose los diferentes tipos y modelos de éstos. Éste capítulo tiene como finalidad mostrar las perspectivas que ofrecen los

sistemas distribuidos para soportar el trabajo cooperativo.

En el capítulo 3 se presenta la arquitectura del *facilitador*. El diseño de la IC y del SAP se desarrolla en el capítulo 4. Y en el 5 se da la especificación de la IC.
Finalmente, en el Apédice 1. se presenta la implantación de la IC y del SAP.

Capítulo 1

Sistemas Cooperativos

1.1 Introducción

Los sistemas cooperativos forman parte del área de Inteligencia Artificial Distribuida (IAD). El objetivo de estos sistemas es que los distintos entes (llamados agentes), por los cuales están conformados, cooperen entre sí para lograr un objetivo común, o para evitar el traslape de metas.

Las ventajas que ofrecen este tipo de sistemas son las mismas que ofrecen los grupos humanos de trabajo, esto es, una mas rápida y exacta o correcta solución a un problema dado, y además ofrecen una mejor opción a la resolución de problemas inherentemente distribuidos.

Se han desarrollado algunos mecanismos para lograr la cooperación, estos son: Red de contratos, Pizarrón, Organizaciones y Modelos. Cada uno de estos esquemas tiene características diferentes, sin embargo, todos ellos persiguen un mismo objetivo, la de lograr la cooperación entre los agentes para alcanzar un fin común.

La idea de sistemas cooperativos esta ya presente en la concepción de Minsky [BG88], de la mente como una sociedad de agentes cooperativos y en la concepción de Arbib del procesamiento de información por el cerebro como una colección de "esquemas" concurrentes.

cooperación de sistemas expertos fue planteada por primera vez por Lesser y Erman con el desarrollo del sistema de entendimiento de habla llamado Hearsay-II [LE80].

Cuando hablamos de sistemas cooperativos siempre se encuentran presentes elementos tales como: agentes múltiples (2 o más), interacción entre los agentes, objetivo particular de cada agente, y objetivo de los agentes en conjunto. Este último elemento es el que determina o motiva a que ocurra lo que podemos llamar un comportamineto cooperativo.

En esta sección el elemento central es el agente, considerado como el único objeto activo, el cual tiene habilidades tales como: percepción, razonamiento y actuación; estas habilidades se refuerzan con la capacidad de representar conocimiento, de razonar (elaborar inferencias) y la habilidad de comunicarse con otros agentes.

La interacción entre los agentes se refiere a la comunicación que se establece entre ellos. Este elemento es fundamental en un sistema cooperativo ya que, como se mostrará más adelante, es la comunicación entre los agentes la que permite que un conjunto de objetivos particulares se conviertan en un objetivo común (o global).

Analicemos ahora los distintos esquemas en los cuales existe la cooperación [WSJ88]. Consideremos primero el esquema en donde los objetivos particulares de los agentes son iguales. Ilustremos este esquema con un ejemplo: Se tiene un pay y dos niños (2 agentes), para los cuales el objetivo particular de cada uno es obtener un pay. Existe cooperación cuando:

1. Platican acerca de esto y lo parten a la mitad.

2. Platican acerca de esto y uno de ellos decide no comer pay si su compañero le puede ofrecer mañana.

3. Platican acerca de esto y uno de ellos propone al otro darle pastel si le da el pay, y como al otro le gusta más el pastel entonces acepta.

En el primer escenario tenemos comunicación entre los agentes y cada uno alcanza su objetivo particular. En el segundo escenario hay

comunicación entre los agentes, sin embargo, sólo uno de ellos alcanza su objetivo. Esto significa que puede haber cooperación aunque los objetivos de los agentes no sean alcanzados. En los dos escenarios se forma un objetivo global. En el primer escenario el objetivo global es partir el pay a la mitad para que cada uno se quede con una parte. En el segundo escenario el objetivo global es que uno de los agentes se quede con el pay a condición de que el otro pueda obtener una parte al día siguiente.

El objetivo global no es más que la formulación de un acuerdo, comunicando las "necesidades particulares" de cada agente, sin embargo, a veces alguno de los agentes tendrá que "ceder" (como en el segundo escenario), es decir, posponer su objetivo particular, o de plano "olvidarlo" cambiándolo por el cumplimiento de otro objetivo particular de mayor importancia, como sucede en el escenario 3.

Todo esto es un proceso de negociación donde la comunicación es un elemento fundamental, tanto para comunicar primero las necesidades particulares, segundo las propuestas y tercero la formulación de un plan para lograr el objetivo común que se haya acordado. Un plan no es más que el ordenamiento de una secuencia de actividades que se llevarán a cabo para alcanzar el objetivo común.

No existe cooperación cuando:

1. Se pelean por el pay y uno se queda con este
2. Se pelean por este partiéndolo a la mitad
3. Su mamá les da a cada uno un pay

En los dos primeros escenarios sólo hay comunicación de las necesidades particulares de cada agente, sin embargo, no hay comunicación para llegar a algún acuerdo, y por tanto no se llega a la elaboración de un objetivo común.

En el segundo escenario aunque los 2 alcanzan su objetivo particular no hay cooperación en ningún momento. En el tercer escenario puede ni siquiera haber comunicación de las necesidades particulares entre los

niños, menos aún habrá comunicación para establecer algún acuerdo. En los tres escenarios la comunicación es muy pobre o nula y por lo mismo no existe negociación ni la formulación de un acuerdo común.

Consideremos ahora el escenario en donde los objetivos particulares de los agentes no son compartidos. Ilustremos este esquema con un ejemplo:

Una niña quiere un pay. Un adulto quiere un café. Discuten cual será el mejor lugar para que los dos compren. Deciden ir a una cafetería. Discuten por que medio llegar a este lugar y deciden que la mejor opción es tomar un autobús. Llegan a la cafetería y compran lo que querían.

En este escenario hay comunicacion de las necesidades particulares de cada agente, de las propuestas para llegar a un objetivo común y de cual será la estrategia (el plan) para lograr este último. Existe aquí un proceso de negociacion obteniéndose de este un objetivo común (ir a una cafetería), aunque los objetivos particulares son diferentes (la niña quiere un pay y el adulto un café). A partir de este objetivo común se formula un plan(tomar un autobús).

1.2 Razones de la existencia de la IAD

Hay varios motivos para distribuir inteligencia, como por ejemplo por razones geográficas (físicas), es decir, el sistema que queremos representar es Inherentemente distribuido lógica o geográficamente. Otra causa es la distribución natural funcional de un problema, como por ejemplo tenemos un sistema operativo donde cada entidad como es el "file system", el "kernel" y el "buffer" entre otros, tienen funciones diferentes También se puede requerir de la distribución del control de un sistema.

Algunas de las razones principales para distribuir un sistema de IA son las siguientes [BG88]:

- Adaptabilidad. El hecho de que un sistema sea distribuido provee gran adaptabilidad a cambios, ya que si falla algún elemento, otro puede sustituir a este, cosa que no sucede en un sistema centralizado.

- Costo. El costo puede ser menor en un sistema distribuido si se utiliza un sistema de cómputo de bajo costo.
- Desarrollo y manejo. En un sistema distribuido cada parte puede ser construida independientemente de las demás. Y además puede crecer, aumentando los módulos que se requieran.
- Velocidad y eficiencia. La concurrencia y el paralelismo pueden incrementar significativamente la velocidad de cómputo y de razonamiento.
- Aislamiento/autonomía. Provee protección a las partes que estén separadas y aisladas unas de otras.
- Confiabilidad. Los sistemas distribuidos pueden ser más confiables que los sistemas centralizados, esto debido a la redundancia que proveen y a su degradación gradual.
- Limitación de recursos. Cada agente computacional tiene, una racionalidad limitada, es decir, no lo puede calcular y razonar todo, ni siempre podrá realizarlo en el tiempo que lo requiera.
- Especialización. Se pueden hacer módulos especialistas en alguna parte del dominio del problema.

En un sistema distribuido de Inteligencia Artificial los agentes pueden tener un diferente "foco de atención" en el dominio del problema, de manera que los agentes tienen una percepción diferente de los objetos y eventos. Los agentes también pueden tener un "conocimiento" diferente al de los demás y debido a las percepciones y conocimientos diferentes de un agente pueden surgir "interpretaciones" diferentes de los objetos y eventos. Debido a esto los agentes pueden poner diferentes valores a los recursos. Los agentes pueden tener diferentes recursos y por lo mismo diferentes niveles de confiabilidad, de igual manera los agentes pueden tener diferentes niveles de "autoridad" y de "credibilidad". En la figura 1.1 se muestra un esquema que resume lo anterior.

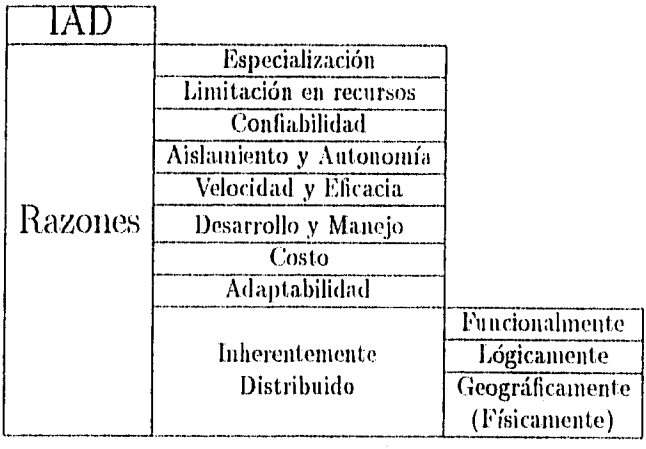


Fig. 1.1 Razones de la existencia de la IAD

1.3 Problemas básicos de la IAD

Los primeros problemas que surgen en un sistema IAD son: la formulación, descripción, descomposición, y asignación de tareas [BG88]. Posteriormente se tiene la solución de los subproblemas y por último la integración de resultados. La formulación y descripción requieren de la representación del problema así como la decisión de los límites de éste. La descomposición de tareas consiste en la partición de una tarea en subtareas más pequeñas y fáciles de resolver.

Con la solución de tareas se pretende proporcionar cada tarea al agente disponible que mejor la pueda atender, esto depende de la habilidad de los agentes para comunicarse e interactuar unos con otros. Algunos otros problemas son los de asegurar que los agentes actúen de manera coordinada y coherente, ya que coordinación no implica necesariamente cooperación, debido a que agentes antagonistas pueden estar coordinados en procedimientos legales. Es decir, la cooperación es un caso particular de la coordinación donde los agentes no son antagonistas, de esta manera, una buena coordinación conduce a una buena coherencia.

Y por último capacitar a los agentes para representar y razonar acerca de acciones, planes y conocimiento de otros agentes para coordinarse con ellos, así como reconocer y reconciliar puntos de diferencias; y resumir los resultados obtenidos son problemas que tienen que ser resueltos en un sistema de IAD. En la figura 1.2 se muestra un esquema que resume lo anterior.

IAD	
Problemas Básicos	Formulación
	Descripción
	Descomposición
	Asignación de Tareas
	Comunicación entre agentes
	Coherencia
	Representar Conocimiento
	Razonar
	Reconocer y reconciliar diferencias
	Resumir resultados

Fig. 1.2 Problemas básicos de la IAD

1.4 Mecanismos para la cooperación

La solución cooperativa de problemas se puede apoyar en uno o más mecanismos de actuación. A continuación se presentan una serie de mecanismos que han sido desarrollados para lograr una solución cooperativa a un problema.

1.4.1 Red de Contratos

El esquema de la Red de Contratos [DS88] parte de la suposición de que dada una tarea, ésta ya ha sido descompuesta en varias subtareas. La red de contratos se encarga entonces de llevar a cabo la distribución de las subtareas y de recibir las subsoluciones. Se tiene entonces que el administrador (el que va a repartir las tareas) manda un mensaje de anuncio de tarea a los nodos. El envío del mensaje puede ser de 3 tipos:

Broadcast (para todos los nodos), multicast (para un grupo específico de nodos), y monocast (dirigido a un solo nodo). Posteriormente cada uno de los nodos que recibe el mensaje hace una evaluación local, es decir, determina si es capaz y si está disponible para realizar la tarea. Puede no ser capaz en el caso de que no tenga la habilidad para realizar la tarea o no estar disponible si su carga de trabajo es excesiva. Una vez relizada la evaluación local, los nodos que están dispuestos a llevar a cabo la tarea, se lo hacen saber al nodo administrador mandándole un mensaje. El administrador hace una evaluación de las propuestas recibidas y elige a uno o varios candidatos mandándole un mensaje para que éstos se convierlan en contratantes. El contratante manda un reporte al administrador para informarle que la tarea ha sido parcial o totalmente ejecutada. El administrador puede terminar el contrato con un mensaje de terminación.

1.4.2 Esquema de Pizarrón

En este esquema existe una estructura llamada pizarrón [LE80] la cual es una área de trabajo. Dicho pizarrón es dividido en varios niveles, en donde pueden leer o escribir varias fuentes de conocimiento (FC). Cada FC es un especialista en una parte del dominio del problema y de igual manera, cada nivel de pizarrón representa una parte del dominio del problema.

Los niveles inferiores del pizarrón son de menor abstracción que los niveles más altos. Las FC "escriben" hipótesis en el pizarrón a las cuales le asocian un grado de credibilidad.

Una fuente de conocimiento se activa cuando se cumple alguno de sus patrones por alguna de sus estructuras de hipótesis en el pizarrón. En este momento las FC entran en la cola del Despachador de Colas (Schedulling Queues) quien junto con la Base de Datos de Foco de Control (Focus of Control Data Base) decidirá cuando ejecutará una acción de escritura sobre el pizarrón.

El Despachador (Squeduller) calcula el nivel de prioridad por cada activación en espera, de acuerdo al grado de impacto que tendrá la información que se genere. La Base de Datos de Foco de Control tiene

meta-información acerca de la actividad para solucionar el problema en cuestión. La meta-información se refiere a cosas como: la mejor hipótesis que se encuentra en el pizarrón, cuanto tiempo ha pasado desde que estas hipótesis fueron generadas o combinadas con otras.

El Monitor se da cuenta cuando alguna de las FC se activa y lo registra en el Despachador de Colas, así como en la Base de Datos del Foco de Control. El procesamiento es distribuido ya que cada FC está encapsulada y es independiente de las demás. El control es distribuido ya que la activación de cada FC está basada en la generación y modificación de hipótesis en el pizarrón. Sin embargo, el Despachador, el Monitor, el Despachador de colas, y la Base de Datos del Foco de Control son centralizados. Esta arquitectura fue implementada, por primera vez, para un sistema de entendimiento de habla llamado Hearsay [LE80]. El sistema simula una red, en la cual cada nodo tiene una estructura de pizarrón, aunque las FC son las mismas, ya que cada nodo tiene un "área de interés" que delimita la porción del máximo rango del pizarrón que es accesada por las FC de ese nodo.

Cuando alguna FC escribe una hipótesis en el pizarrón de su nodo, esta modificación es comunicada a los demás nodos para que actualicen la información. Los nodos cooperan al generar e intercambiar soluciones parciales basadas en puntos de vista locales. Al estar iterativamente intercambiando los nodos sus soluciones parciales, eventualmente convergen a una solución global.

1.4.3 Organizaciones

Un sistema cooperativo tiene que tener una estructura organizacional que permita determinar o guiar el esquema de cooperación entre los agentes para la solución conjunta de un problema.

Organizaciones según Fox

Podemos ver a un sistema distribuido como una organización en particular que se encarga de la descomposición de tareas y del régimen de control [S.80].

Se requiere de una estructura organizacional para que un sistema cooperativo pueda funcionar como tal, y se requiere de este último para romper las barreras impuestas por la "racionalidad limitada". La teoria de la racionalidad limitada de Simon [S.80], expresa que la capacidad de la mente humana está limitada para resolver los problemas del mundo real. Y lo mismo puede decirse de una computadora, ya que esta tiene un límite en la velocidad de su procesamiento, y en la capacidad para almacenar datos.

La racionalidad limitada es un elemento que encontramos tanto en seres humanos como en sistemas de cómputo, y en los dos ésta da cabida a que se formen sistemas cooperativos controlados por organizaciones para solucionar esta limitante.

De aquí podemos darnos cuenta que las organizaciones no sólo pueden ser aplicadas a grupos de seres humanos, sino también a sistemas de cómputo. Muchas teorías y conceptos han sido desarrollados en el tema de organizaciones de grupos humanos, pues bien, toda esa teoría puede ser reutilizada y aprovechada para sistemas de cómputo.

Fox destaca la existencia de un proceso evolutivo en las organizaciones [S.80], el cual se muestra en la figura 1.3.

La organización más simple es una persona, esto es cuando ésta es capaz de solucionar todos los problemas que se le presentan. Sin embargo, cuando surge un problema de tal magnitud que esta sola persona no pueda resolver, ya que su racionalidad limitada se lo impide, entonces se requiere de la formación de un grupo de dos o más personas. En este momento inicia la cooperación, los elementos del grupo deben formar un objetivo común y repartir cada tarea al elemento más adecuado para que la lleve a cabo. La comunicación entre los elementos del grupo juega un papel muy importante en el desarrollo del trabajo cooperativo.

Cuando la dimensión del grupo es considerablemente grande es conveniente establecer una jerarquía simple, ya que el costo de la comunicación y de la distribución de la información se incrementa mucho. Así surge la jerarquía simple bajo dos niveles. El más alto nivel tiene un elemento que se encarga de la toma de decisiones, y se encarga de

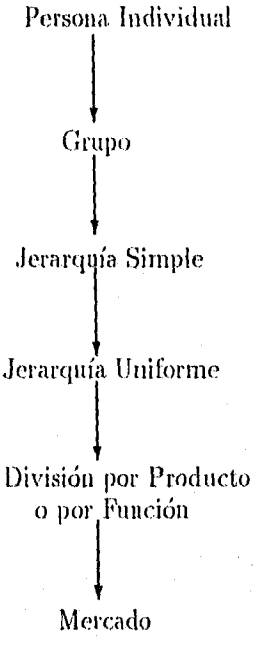


Fig. 1.3 Proceso evolutivo de las organizaciones

coordinar a los que se encuentran en el nivel inferior.

Toda esta información debe de estar disponible, sólo para el elemento del primer nivel y es que éste es el que autoriza cualquier acción o movimiento de información.

En la medida en que las jerarquías simples crecen, estas se vuelven insuficientes para procesar toda la información, y entonces se requiere de jerarquías uniformes, las cuales tiene múltiples niveles de toma de decisiones. Cada nivel funciona como un filtro para su nivel superior.

Al incrementar de tamaño las jerarquías uniformes obtenemos una competencia por los recursos compartidos debido a los múltiples productos producidos. Este problema se puede resolver estableciendo una organización que es dividida por líneas de producción o de funciones. En este esquema, a diferencia de los demás, el control es local y la información que habilita el control está disponible.

La organización es particionada por producto como por ejemplo cuando tenemos una fábrica de coches la cual se divide en varias fábricas, una para hacer puertas, otra para hacer llantas, otra para hacer motores, etc.

En la organización funcional tenemos por ejemplo la organización de un sistema operativo del cual forman parte el despachador, el sistema de archivos, el controlador de E/S etc., y todos ellos describen una parte funcional del sistema.

No obstante, al crecer la organización de división por producto o por función, también crecen los problemas de coordinación así como la cantidad de información que requiere ser procesada. Estos dos últimos pueden ser disminuidos al utilizar una organización de mercado, el cual es un sistema de precios en donde los recursos son utilizados sin pérdidas, ya que se establece una competencia en donde gana el que vende a mejor precio y tiene los gastos menores.

Organizaciones según Mallone

Mallone [Tho86] describe una serie de estructuras de coordinación que modelan ciertas clases de procesamiento de información de estructuras organizacionales. Una estructura de coordinación está definida como un patrón de toma de decisiones y comunicación entre un conjunto de agentes que ejecutan ciertas tareas para lograr determinados objetivos [Tho86].

Las estructuras de comunicación propuestas por Mallone son: jerarquía de productos, jerarquía funcional, mercado descentralizado, mercado centralizado.

En la jerarquía por producto tenemos algo muy similar a la divsión por producto descrita por Fox. En este tipo de estructura hay divisiones separadas para las diferentes líneas de productos. Cada división tiene un "administrador de producto". La General Motors es un buen ejemplo de este tipo de estructura ya que tiene una división que se encarga de fabricar los Chevrolet, otra los Pontiac, otra los Cadillac, etc. El administrador de producto decide que tareas necesitan ser realizadas para producir un producto (para alcanzar algún objetivo), y asignar estas a los procesadores (agentes) capaces de llevarlos a cabo.

La comunicación sólo se establece entre cada administrador de producto y sus procesadores. De manera que todos los eventos que suceden en una división no pueden afectar a otra división, ya que son totalmente independientes.

La jerarquía funcional es similar a la organización por división de función descrita por Fox. En esta estructura, procesadores de un mismo tipo son agrupados en un mismo departamentos el cual es administrado por el "administrador funcional". Y los departamentos son administrados por el ejecutivo de oficina", el cual decide qué tareas necesitan hacerse para producir todos los productos de una organización. Cuando se requiere asignar una tarea, el "ejecutivo de oficina" entrega la tarea al "administrador funcional" del tipo adecuado, y éste a su vez, manda a hacer la tarea de sus procesadores. La decisión que toma el administrador funcional al asignar alguno de sus procesadores una tarea la

toma en base a la carga de trabajo que estos tengan.

1.4.4 Modelos

Mace (Multi-Agent Computing Environment) [LCN88] es una herramienta para simular el comportamiento de un sistema distribuido. Los elementos del sistema Mace son los siguientes:

- Un conjunto de agentes.
- Una comunidad de agentes del sistema. Estos se encargan de la interpretación de comandos, interfaces a ususarios, monitor de ejecución, manejo de errores, etc.
- Una base de datos descriptiva. Esta contiene una descripción de los agentes.
- Un conjunto de Kernels. Un Kernel se encarga del manejo del ruteo de mensajes, ejecuta E/S a terminales, archivos y otros dispositivos, y coordina la ejecución de los agentes.

Los agentes tienen conocimiento de sí mismos y de otros agentes. En base a esto, es que ellos se coordinan para resolver algún problema. Los agentes necesitan saber información de los agentes tal como: su identidad, dirección, habilidades, etc. Para representar toda esta información los agentes utilizan modelos, esto es, un agente "modela en el mundo" a los agentes de los cuales tiene información. Los agentes se organizan en sub-unidades o coaliciones al trabajar conjuntamente en la resolución de un problema.

Un agente en Mace es un objeto activo que se comunica usando mensajes. Estos existen en un medio ambiente, el cual son capaces de percibir. Un agente tiene atributos y es capaz de tomar acciones.

El conocimiento que tiene un agente está albergado en un atributo llamado "Conocido" (acquaintance). "Conocido" es una base de datos asociativa que provee un modelo ambiental utilizando modelos explícitos de otros agentes en el mundo. Un modelo del mundo tiene representación de otros agentes en base a los siguientes calificadores:

- Nombre. Nombre del agente modelado.
- Clase. Nombre de la clase del agente modelado.
- Dirección. La dirección del agente modelado.
- Rol. La(s) relacione(s) que el agente modelado tiene con el agente que lo modela. El rol tiene cuatro valores predefinidos:
  1. self.- el modelo es del agente por sí mismo.
  2. creador.- el modelo es del creador del agente.
  3. org-miembro.- el modelo es de un miembro de la organización que este agente define.
  4. miembro-de.- el modelo es una abstracción que es parte de una organización padre.
- Habilidades. Son las habilidades que este agente conoce del agente modelado. Las habilidades son pares que consisten de los objetivos que la habilidad alcanzará, y de un conjunto de procedimientos alternativos para alcanzar los objetivos. Es aquí donde se define un plan, al establecer una secuencia de actividades a seguir para alcanzar un objetivo. Sin embargo el plan es siempre el mismo, es decir, no cambia.

En el proceso de resolución de un problema, cada agente construye un modelo del mundo y en base a éste construye un plan y tomando todo lo anterior se forman organizaciones jerárquicas en donde la habilidad de la organización, la cual tiene asociada un objetivo, está definida por el conjunto de las habilidades individuales, asociadas a su objetivo individual de cada elemento de ésta.

En esta organización cada elemento en la escala jerárquica se ve afectado por la racionalidad limitada para el cumplimiento de sus objetivos. Ya que un elemento para poder alcanzar sus objetivos requiere de la ayuda de una sub-organización y esta a su vez puede requerir ayuda de una de sus sub-organizaciones.

1.5 Conclusiones

Los sistemas cooperativos ofrecen una buena alternativa para resolver problemas que son física, lógica o funcionalmente distribuidos, además de que son aptos para resolver el problema de la limitación de recursos (razón limitada) e incrementar la confiabilidad, aislamiento y autonomía, la velocidad y eficiencia, la adaptabilidad así como para disminuir los costos. La solución cooperativa de problemas puede ser llevada a cabo con uno o más mecanismos para la cooperación. Estos son: la Red de Contratos, el esquema de Pizarrón, Organizaciones y Modelos.

Todos estos esquemas de cooperación requieren ser sustentados por un sistema que soporte la interacción entre los agentes, los cuales se encuentran y desenvuelven en un ambiente distribuido. Se requiere entonces de un sistema distribuido como plataforma para la integración de un sistema de IAD, mismo que nos permite construir aplicaciones cooperativas; formándose así una jerarquía de plataformas como se ilustra en la figura 1.4, donde la capa inferior soporta a la superior. En la figura 1.5 se muestra un esquema que resume lo anterior.

	Red de Contratos	Pizarrón 1	pizarrón 2	Organización	Modelos
Negociación	Si hay	No hay	No hay	Si hay	No hay
Organización	Dinámica implícita	Estática Implícita	Estática Implícita	Dinámica y estática Explícita	Estática Explícita
Planeación	No hay	Dinámica Abstracción media	Dinámica Abstracción alta	No hay	Estática Abstracción media
Control	Distribuido	Distribuido	Distribuido	Distribuido y centralizado	Distribuido
Complejidad	Sencillo	Medio	Medio a Complejo	Sencillo a complejo	Sencillo a complejo
Racionalidad limitada	Implícita	Implícita	Implícita	Explícita	Implícita
Patrón de Interacción	Negociación según capacidad y disposición	Priorización de actividades en base a comunicación	Priorización de actividades en basu a comunicación	Evolutivo en base a la racionalidad limitada	Evolutivo en base a la racionalidad limitada
Toma de acción	Nivel tarea	Individual y cooperativo	Individual y cooperativo	Individual y cooperativo	Individual y cooperativo
Rango de Organización	Dispersión por negaciación	Predictiva	Predictiva	Estructural	Estructural
Roles sociales	No hay	No hay	No hay	Si hay	Si hay

Fig. 1.5 Evaluación de los mecanismos para la cooperación

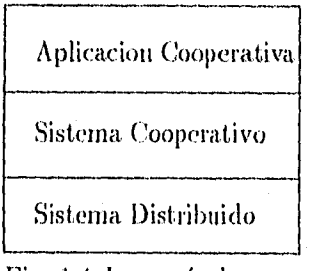


Fig. 1.4 Jerarquía de capas en un sistema de IAD

Capítulo 2

Sistemas Distribuidos

2.1 Introducción

Un sistema de IAD requiere de un sistema distribuido (SD) como plataforma para que pueda operar. Dentro de los SD existen diversas formas de comunicar los elementos que lo conforman. De la misma manera existen diferentes lenguajes y modelos para construir un SD. La pregunta que surge es la siguiente: ¿Qué mecanismos de comunicación y qué modelos son los más adecuados para construir una plataforma para un sistema de IAD?

Cualquier SD tiene (o cuando menos busca tener) las propiedades de paralelismo y de tolerancia a fallas. Además de esto un SD ofrece varias ventajas sobre los sistemas centralizados, como son:

- mejor resolución a problemas inherentemente distribuidos
- evitar que la falla de un elemento del sistema provoque la falla en todo el sistema
- una más rápida resolución a un problema, debido a la cooperación de varios elementos para resolverlo
- modularización del sistema
- mayor confiabilidad

En esta sección se ilustra lo que es un sistema distribuido, sus mecanismos de comunicación, los modelos y lenguajes concurrentes, y las características deseables para construir una plataforma adecuada para un sistema de IAD.

2.2 Características de los SD

Existe una gran diversidad de definiciones en la literatura de lo que es un sistema distribuido, y éstas presentan considerables desacuerdos. Sin embargo, entre todos los puntos discordantes, existe uno en el que todos están de acuerdo: se requiere la presencia de multiples procesadores [BSS91].

La gran variedad de concepciones de lo que es un sistema distribuido se debe a la gran diversidad de arquitecturas de sistemas de procesadores múltiples. Existen por ejemplo computadoras vectoriales, como la Cray, que tiene varios procesadores que paralelizan operaciones aritméticas, como las que se realizan con matrices. Las máquinas multiprocesador tienen varios procesadores, pero todas ellas comparten una misma memoria. Las multicomputadoras tienen varios procesadores, pero a diferencia de las anteriores, estas no comparten una misma memoria. Como ejemplo de las últimas tenemos las redes de computadoras y las de transputers.

Sobre todos estos tipos de arquitecturas se tiene desacuerdo entre los expertos de si deben considerarse o no sistemas distribuidos. Entre una variada divergencia de concepciones se ha elegido una definición para identificar lo que un sistema distribuido es [BSS91]: *Un sistema distribuido de cómputo consiste de múltiples procesadores autónomos que no comparten memoria principal, pero que cooperan mandando mensajes a través de la red de comunicaciones.*

Los procesadores ejecutan procesos, donde un proceso es una instancia de un programa en ejecución. La programación concurrente es la actividad de construir un programa que contenga múltiples procesos, los cuales cooperan para realizar alguna tarea [R.91].

Dentro de la programación concurrente existen tres clases de procesamiento: multiprogramación, multiprocesamiento y procesamiento distribuido [RB83]. En la multiprogramación los procesos se ejecutan concurrentemente compartiendo uno o más procesadores, y estos comparten la misma memoria; por lo tanto la multiprogramación no se realiza en una arquitectura con un sistema distribuido. El multiprocesamiento es igual que la anterior, sólo que a cada proceso le corresponde un procesador, sin embargo, tampoco este sistema se ejecuta en una arquitectura distribuida. Sólo el procesamiento distribuido cumple con la arquitectura de un sistema distribuido. Aquí los procesos que se ejecutan concurrentemente les corresponde un procesador, y cada uno de estos no comparten la misma memoria.

La programación paralela coincide con los casos de multiprocesamiento y la de procesamiento distribuido. Éste tipo de programación requiere la posibilidad de que varios procesos se ejecuten al mismo tiempo, cada uno en un procesador, para trabajar una tarea en forma conjunta.

Se dice que un programa es seudoparalelo si los procesos que los componen se ejecutan de manera conjunta en varios procesadores, pero sin embargo, cuando menos dos de los procesos comparten un procesador.

La granularidad de los programas paralelos es de dos tipos: gruesa y fina [BSS91]. En la gruesa los programas pasan la mayoría del tiempo haciendo cálculos y su comunicación es reducida. Los de granularidad fina se comunican frecuentemente.

Existen cuatro tipos básicos de procesos en un sistema distribuido: filtros, clientes, servidores, y pares(peers) [R.91]. Un proceso filtro recibe los datos de entrada de otro proceso, y el primero realiza un filtrado de estos. En Unix un proceso *sort* es un filtro. Por ejemplo el comando *who* | *sort* crea dos procesos, *who* y *sort*, el primero le entrega al segundo el nombre de los usuarios que se encuentran en sesión y el segundo los ordena alfabéticamente.
Los procesos clientes trabajan en conjunto con los servidores. Los segundos ofrecen servicios y los primeros lo solicitan. Estos usan el mod-

elo cliente/servidor.
Un ejemplo de un servidor es el proceso *in.telnetd* de Unix que ofrece el servicio de proporcionar una sesión remota, y el cliente es el proceso *telnet* que solicita dicha sesión.

Los procesos pares, son procesos repetidos; el objetivo de estos es realizar una tarea repetida. Como ejemplo tenemos un sistema de archivos repetidos, es decir, cada archivo se encuentra repetido uno o más veces.
Para cada archivo existe un proceso para realizar operaciones sobre éste. Por ejemplo, cuando un archivo es abierto el proceso par de este se lo informa a los demás pares para que ejecuten la misma acción sobre su respectivo archivo.

2.3 Comunicación entre procesos

Para que los procesos en un sistema distribuido cooperen, se requieren dos elementos: comunicación y sincronización [RB83].
El primero es necesario para que los procesos puedan conocer el estado y requerimientos de otros. La sincronización es indispensable para que los procesos se comuniquen. Esta representa un conjunto de restricciones para el ordenamiento de eventos. Mediante un mecanismo de sincronización se retraza la ejecución de un proceso para satisfacer las restricciones.
Por ejemplo; se tiene el evento e_x que consiste en que el proceso A mande un mensaje al proceso B; y el evento e_y el cual es que el proceso B reciba un mensaje del proceso A.
De manera que debe haber un mecanismo de sincronización que retarde e_y hasta que ocurra e_x. Ya que e_y no tiene sentido si no ha sucedido e_x.

Los tipos de comunicación pueden ser: paso de mensajes asíncrono, paso de mensajes síncrono, comunicación generativa, Remote Procedure Call (RPC, con sus siglas en inglés) o llamadas a procedimientos remotos, y Rendezvous [R.91].

La comunicación se realiza a través de canales y estos pueden ser

de tres tipos: directos, globales (o mailboxes) y puertos [RB83]. En los primeros se establece una comunicación de uno a uno, en los segundos de muchos a muchos y en los últimos de muchos a uno.
Los canales pueden ser estáticos o dinámicos [RB83]. Un proceso que tiene un canal estático no lo puede cambiar, de modo que la comunicación siempre la dirigirá hacia ese canal. Los canales dinámicos permiten cambiar los canales.

2.3.1 Paso síncrono y asíncrono de mensajes

En estos tipos de comunicación existen dos tipos de primitivas básicas: el *send* y el *receive*.
El proceso que transmite un mensaje utiliza *send*. El que recibe un mensaje utiliza *receive*, y una vez que utiliza esta primitiva queda bloqueado hasta que el proceso transmisor ejecute *send*.

Con el paso síncrono de mensajes el proceso transmisor espera a que el proceso receptor reciba el mensaje. En cambio, con el paso asíncrono de mensajes el trasmisor no espera a que el receptor reciba el mensaje, sino que continúa su ejecución.
Se dice que el asíncrono tiene ventaja sobre el síncrono en el sentido de que el primero al no bloquearse nunca, el transmisor no limita el paralelismo del programa. Sin embargo, el segundo permite la sincronización de los dos procesos.

2.3.2 Comunicación Generativa

En este tipo de comunicación [D.85] todos los mensajes son mandados a un lugar llamado "espacio de tuplos", e igualmente todos los mensajes son leídos de ese lugar.
Ésta es similar al paso de mensajes asíncrona desde el punto de vista de que el proceso transmisor nunca se bloquea.
Una ventaja de este tipo de comunicación es que el transmisor no necesita conocer al proceso receptor, sino que el espacio de tuplos se encarga de entregar los mensajes al receptor adecuado.
Pareciera que funciona así, aunque, en realidad los receptores se encargan de buscar los mensajes en el espacio de tuplos.

2.3.3 RPC y Rendezvous

El Remote Procedure Call (RPC, con sus siglas en inglés) o llamada a procedientos remotos, y Rendezvous son primitivas de comunicación síncrona, que permite que el flujo de información sea en doble sentido. Y debido a lo anterior, estas dos se adecúan mucho al modelo cliente/servidor, ya que en éste el flujo de mensajes siempre va en dos sentidos.

El modelo rendezvous se basa en tres conceptos [BSS91]: la declaración de entrada (entry declaration), la llamada de entrada(entry call), y la sentencia de aceptación (accept statement).
La primera y la última son utilizadas por el servidor y la segunda por el cliente.
Una declaración de entrada es similar, desde el punto de vista sintáctico, a una declaración de un procedimiento. La llamada de entrada es similar a una sentencia de un llamado a un procedimiento.
La sentencia de aceptación contiene una lista de sentencias, a ser ejecutadas cuando la entrada es llamada.
Por ejemplo, si tenemos dos procesos C y S, y C llama a la entrada de S, entonces S ejecuta la sentencia de aceptación. Aquí C queda bloqueado desde que realiza la llamada y hasta que S termine de ejecutar la sentencia de aceptación. Los valores de los parámetros de ésta, pueden ser modificados. Al terminar esta sentencia, C queda desbloquedo y S puede seguir trabajando en la solicitud de C.

La RPC es muy similar a un llamado de un procedimiento. Por ejemplo, si se tiene dos procesos C y S, y un procedimiento remoto P en S. Entonces al hacer C el llamado de P, le transmite los valores de los parámetros de este, y se bloquea hasta que S le regrese los valores de los parámetros, esto es, hasta que termine de ejecutar el procedimiento P.

En contraste con rendezvous, el cliente que utiliza RPC se queda bloquedo hasta que el servidor haya terminado de ejecutar su solicitud, mientras que el primero sólo se queda bloqueado mientras el servidor realiza la sentencia de aceptación.

2.4 Características deseables de un SD para IAD

Para pensar en las características deseables de un Sistema Distribuido (SD) que sirva como plataforma de un sistema de IAD, es necesario pensar en los problemas de la última.
Como ya se mencionó en el primer capítulo éstos problemas incluyen:

- Cómo formular, describir, descomponer, y repartir problemas; así como sintetizar los resultados
- Cómo habilitar a los agentes para comunicarse e interaccionar, qué lenguaje de comunicación o protocolos usar, y qué y cuándo comunicar
- Cómo asegurar que los agentes actúen coherentemente
- Cómo habilitar a los agentes individuales para representar y razonar acerca de las acciones planes y conocimiento de otros agentes.
- Cómo reconocer y reconciliar puntos de vista e intenciones conflictivas.

Tener en mente los anteriores problemas nos permite dar una mejor respuesta a la pregunta: ¿Qué arquitectura y cuales son las características que requiere un SD para que soporte y facilite la labor de la IAD?

La programación concurrente basada en objetos es una buena opción para tal plataforma, ya que ésta cuenta con varias características deseables para su construcción. Las razones de esto se darán a continuación.

Las habilidades necesarias para cubrir los requerimientos que necesitan las implementaciones de plataformas para DAI son [LB92]:

- Encapsulación y control local.- Esto provee una base natural para la distribución y una descomposición flexible de tareas.

- Comunicación basada en mensajes.- Se requiere comunicación basada en mensajes en sistemas de IAD para la interacción.

- Objetos de Multigranularidad Heterogénea.- Agentes u objetos, pueden existir a diferentes niveles y tipos de granularidad en sistemas de IAD.

- Soporte del lenguaje para Organizaciones flexibles y estructuras de interacción.- para explotar la concurrencia al máximo, así como para proveer una flexible reconfiguración de las interacciones entre objetos en sistemas de IAD.

- Modelación de agente.- Se requiere de la habilidad de modelar el comportamiento y conocimiento de otros agentes, para reducir la comunicación.

- *Shells* reusables.- Reusar descripciones abstractas de componentes del sistema, reusar bases de conocimiento y estructuras de coordinación.

- Probador y Medidor (herramientas).- Para simular, medir y controlar ambientes

Sin embargo, la Progamación Concurrente Basada en Objetos (PCBO) tiene muy simples los siguientes aspectos [LB92]:

- Comunicación/interacción.- Los mensajes mandados no expresan la intención del mismo.

- Actividad.- La actividad de los objetos es muy procedimental y no declarativa.

- Estado.- Usualmente no hay una descripción explícita del conocimiento del agente.

- Organizaciones.- La IAD necesita la representación de grupos. Se necesitan modelos de organizaciones para estructurar la coordinación entre agentes, esto es, describir como componer/descomponer y repartir tareas entre agentes y lograr una interacción coherente.

2.5 Conclusiones

Los sistemas distribuidos son un elemento indispensable para integrar un sistema de IAD. Dentro de los sistemas distribuidos existen varios paradigmas de comunicación, los cuales son: paso asíncrono de mensajes, paso síncrono de mensajes, comunicación generativa, RPC, y Rendezvous.
Algunos de estos tipos de comunicación son más sencillos e inflexibles, otros son más complejos y flexibles. Cada uno de ellos se adecúa más a un tipo de necesidades de comunicación.
Por ejemplo, si se requiere construir un sistema que corresponda al modelo cliente/servidor, el tipo de primitiva que más se adecúa para este caso son las RPC's. Si se requiere que un sistema utilice mensajes anónimos, debido a que esto facilita su construcción, el más adecuado es utilizar el esquema de comunicación generativa. Pero si se requiere una mayor eficiencia en el paso de mensajes, entonces lo más indicado es el paso síncrono y asíncrono de mensajes.

Por otro lado, se han desarrollado una gran cantidad de modelos y lenguajes distribuidos que se basan en los tipos de comunicación mencionados. Existen otros que se basan en paradigmas diversos como son: basados en objetos, funcionales, lógicos, y procedurales. Todos ellos se adecúan mejor a un tipo de problema.

A pesar de que el modelo de objetos responde de una manera más satisfactoria a las necesidades de la IAD, los lenguajes concurrentes que se han creado bajo este modelo no cubren todos los requerimientos de la IAD, como por ejemplo: los mensajes no expresan la intención, no hay descripción explícita de agente, y tampoco hay facilidades para representar organizaciones.

Capítulo 3

Una red de agentes cooperativos

3.1 Introducción

Como ya se ha mencionado la Inteligencia Artificial Distribuida (IAD) requiere de un Sistema Distribuido (SD) como plataforma para que pueda operar. Dicho SD puede verse como una Red de Agentes Cooperativos (RAC), donde cada uno de los agentes posee capacidades y habilidades para establecer un control del diálogo y de la cooperación que entable con los demás agentes.

Para esto requiere la capacidad para organizarse formando grupos de trabajo. De manera que cada agente pueda pertenecer a varios de éstos. Para ello necesita de ciertas facilidades como la de obtener información de otros agentes.
También es importante que controle varios diálogos a la vez cuando participa en diversos grupos de trabajo.

3.2 La metáfora de la oficina

La RAC está basada en la metáfora de la *oficina*, la cual plantea un trabajo cooperativo mediante el uso de niveles jerárquicos de comunicación.

Una *oficina* está formada por un *jefe* y una *secretaria* los cuales están relacionados de manera jerárquica. Estos dos caracteres tienen disponibles varios servicios de comunicación como: teléfono, fax, y servicio de mensajería.

La *secretaria* le ayuda al *jefe* a organizar el trabajo cooperativo con otros *jefes*, entablando diálogos con otras *secretarias*.
En este modelo existen tres niveles jerárquicos de comunicación: en el nivel *n* se encuentran los *jefes*, en el *n-1* las *secretarias*, y en el *n-2* los servicios de comunicación.
De la misma manera existen tres tipos de diálogos: *jefe-jefe*, *jefe-secretaria*, y *secretaria-secretaria*.

Cada *jefe* se puede comunicar con su *secretaria* utilizando actos de habla simples: órdenes, preguntas, y afirmaciones.
El diálogo que se lleva acabo con las *secretarias* se realiza mediante una gama más amplia de actos de habla relacionados con el proceso de negociación.

Las actividades de las *secretarias* pueden involucrar dos tipos de acciones, como ha sido propuesto por Shoham [Sho90]: *acciones comunicativas* y *acciones privadas*.
Las primeras son la generación e interpretación de actos de habla, y las segundas son las acciones relacionadas al: procesamiento de información, soporte de la toma de decisiones, y acciones físicas sobre el mundo de los agentes.

Las *secretarias* tienen cuatro clases de actividades básicas:

1. Intercambio de mensajes con otros miembros del grupo.
2. Procesamiento de información.
3. Capacidad de toma de decisión.
4. Ejecución física de acciones.

Este modelo servirá como base para la definición de una arquitectura de una red multiagentes. De esta manera asociamos el rol desempeñado

por el *jefe* con la del *usuario*. Donde los *usuarios* pueden ser de dos tipos: activos y pasivos. Los primeros pueden ser seres humanos o sistemas expertos. Los segundos son sistemas de cómputo tales como bases de datos, agendas electrónicas, etc.
En este caso cuando utilizamos la palabra *usuario* nos referimos al *usuario activo*.

El rol de las *secretarias* es asignado a agentes computacionales llamados *facilitadores*, cuya función principal es facilitar el trabajo de sus *usuarios* asociados. Estos son agentes basados en conocimiento y son capaces de cooperar entre ellos.

3.3 Arquitectura del Facilitador

La arquitectura del *facilitador* que a continuación se presenta ha sido desarrollado en el Laboratorio Nacional de Informatica Avanzada (LANIA), ubicado en Xalapa, Veracruz, México.
El *facilitador* es una entidad que permite que un *usuario* coopere con otros, los cuales están conectados a una red de comunicaciones como se muestra en la figura 3.1.
Un *usuario* puede ser un elemento humano o un sistema informático.

Funcionalmente el trabajo cooperativo de un facilitador está dividido en tres módulos, como se indica en la figura 3.2: *máquina de acciones*, *servidor de acciones privadas*, y *máquina de planes*.

La primera se encarga de conmutar las acciones (*comunicativas* y *privadas*) del *usuario*, como se muestra en la figura 3.3. El segundo proporciona diferentes servicios informáticos locales y distribuidos. El tercero es el ejecutor de tareas cooperativas de apoyo al *usuario*. A continuación se analiza con mayor detalle cada uno de los módulos.

3.3.1 La Máquina de acciones

La *máquina de acciones* se encarga de conmutar acciones (*comunicativas* y *privadas*) entre el *usuario* y la *máquina de planes*.
El *servidor de acciones privadas* sólo recibe *acciones privadas* de la

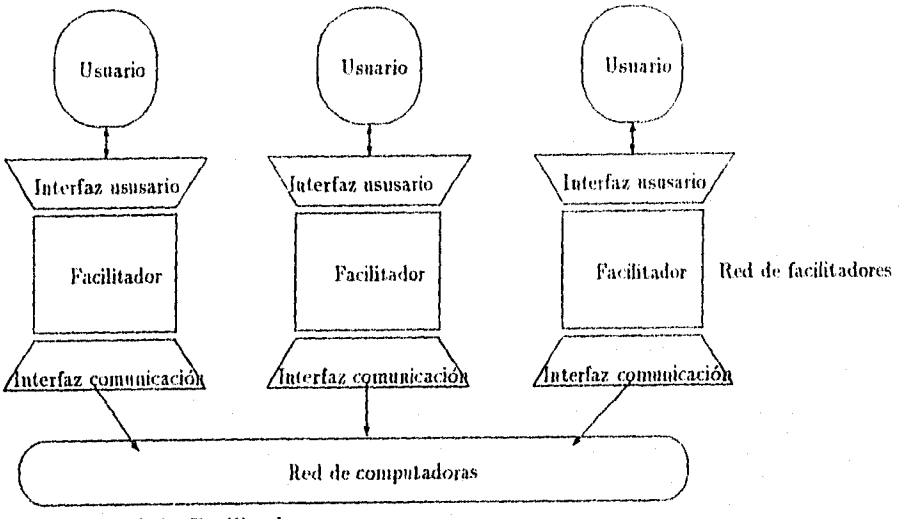


Fig. 3.1 Red de Facilitadores

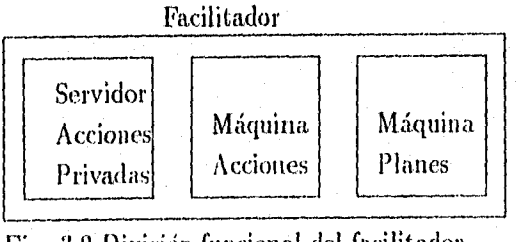


Fig. 3.2 División funcional del facilitador

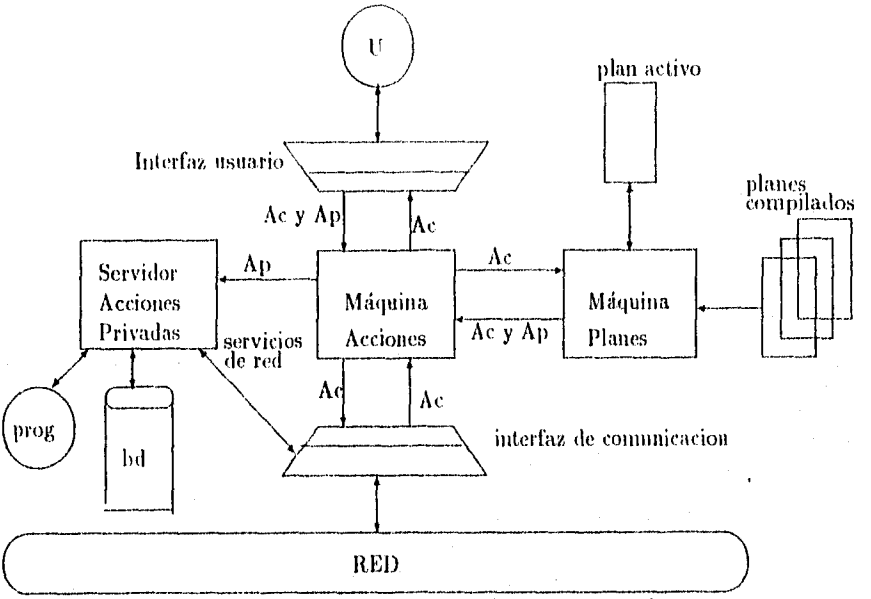


Fig. 3.3 Arquitectura del facilitador

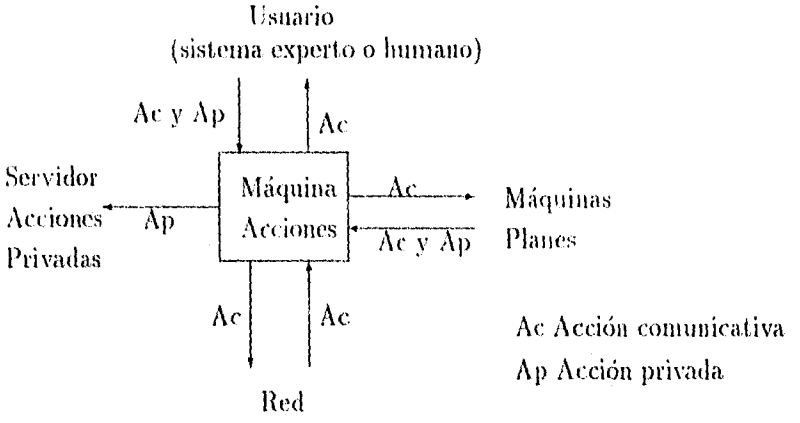


Fig. 3.4 Máquina de Acciones

máquina de acciones.
La *red* nada más puede mandar y recibir *acciones privadas.* Lo anterior se ilustra en la figura 3.4.

Las *acciones comunicativas* (*Ac*) están basadas en actos de habla y permiten el diálogo (intercambio bidireccional de información) entre las siguientes entidades:

- *usuario local* < — — > *usuario remoto*
- *usuario local* < — — > *máquina de planes local*
- *máquina de planes local* < — — > *máquina de planes remoto*
- *máquina de planes remoto* < — — > *usuario local*

La *máquina de acciones* tiene la capacidad de establecer y controlar varios diálogos en forma simultánea.
Las *acciones comunicativas* son solicitadas por el *usuario* y la *máquina de planes*, y son retransmitidas por la *máquina de acciones* y por la *red* con el objetivo de que estas lleguen a su destinatario.

Existen dos tipos de *acciones comunicativas*: elementales y compuestas. Las primeras están formadas por: *req* e *inf.* Las segundas por *pregunta.*

La acción *req* solicita algún servicio, *inf* proporciona información, y *pregunta* solicita alguna información.
Generalmente cuando una entidad utiliza *req* o *pregunta* obtiene como contestación una acción *inf*.
Ésta última puede actuar además de manera aislada, es decir, con el sólo propósito de proporcionar alguna información aunque ésta no haya sido solicitada.

Las *acciones privadas*, son acciones que permiten la realización de un servicio informático local o en red. Éstas son solicitadas por el *usuario* y por la *máquina de planes*.
El *servidor de acciones privadas* es el ejecutor de la acción solicitada.
Algunas *acciones privadas* son:

- Transferencia de archivos.
- Correo electrónico.
- Ejecución de programas locales o remotos.
- Acceso a bases de datos.

3.3.2 La Máquina de planes

La *máquina de planes* se encarga de seleccionar y ejecutar el plan adecuado para llevar a cabo un trabajo cooperativo, como se muestra en la figura 3.5.

Un plan consiste en una secuencia de *acciones privadas* y *comunicativas* organizada en segmentos. Los planes compilados es un conjunto de planes alternativos para una tarea dada.
Un plan es activado por un acto comunicativo proveniente del *usuario* local o de la red (*usuario* o la *máquina de planes remotos*).
La *máquina de planes* selecciona el plan más adecuado para la tarea que le fue encomendada y la ejecuta. Cada acción ejecutada es evaludada, si ésta resulta no ser satisfactoria con respecto al objetivo del plan y si hay alternativas, se selecciona un nuevo plan y se procede a su ejecución.

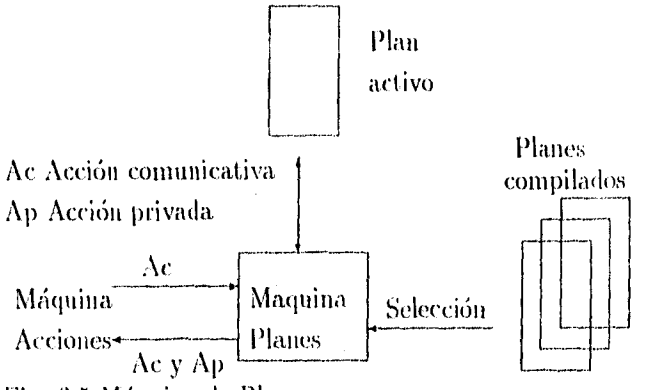


Fig. 3.5 Máquina de Planes

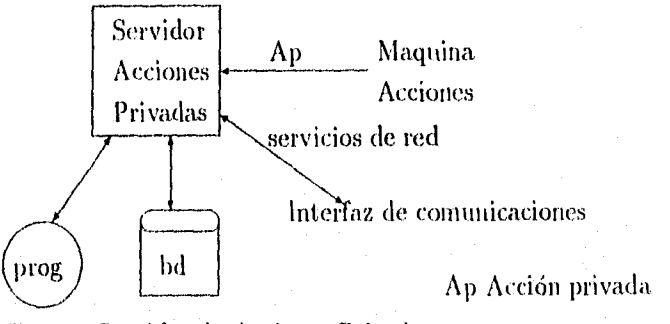


Fig. 3.6 Servidor de Acciones Privadas

3.3.3 El Servidor de acciones privadas

El *servidor de acciones privadas* (SAP) proporciona al *usuario* y a la *máquina de planes*, servicios informáticos.
Las *acciones privadas* que ejecuta este servidor pueden ser locales o remotas, en cuyo caso utiliza los servicios provistos por la *Interfaz de Comunicaciones*. Tal esquema es ilustrado en la figura 3.6.

Los servicios proporcionados localmente pueden ser:

- Ejecución de programas locales.

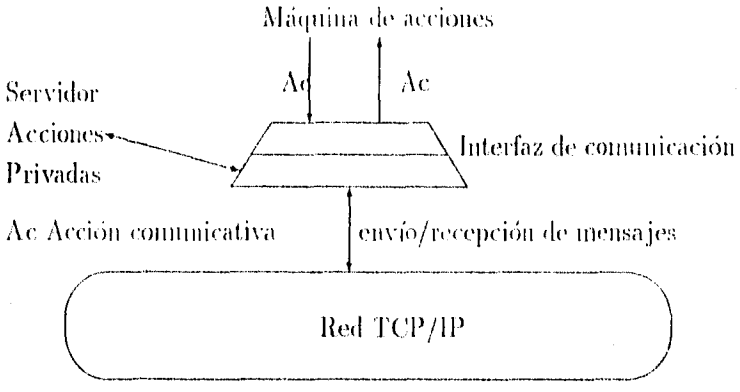


Fig. 3.7 Interfaz de Comunicación

- Acceso a bases de datos locales.

Los servicios proporcionados distribuidamente pueden ser:

- Transferencia de archivos.
- Correo electrónico.
- Acceso a bases de datos distribuidas.
- Ejecución de programas remotos.

3.3.4 La Interfaz de comunicación

La *interfaz de comunicación* (IC) está formada por dos niveles: el primero es una interfaz de acceso a la *máquina de acciones*, y el segundo es una interfaz de acceso a la red de comunicaciones. Lo anterior es ilustrado en la figura 3.7.

El nivel superior proporciona ciertos servicios que permiten a la *máquina de acciones* y al SAP desempeñarse en un trabajo cooperativo. Este nivel provee los siguientes servicios:

- Localización.
- Asociación.

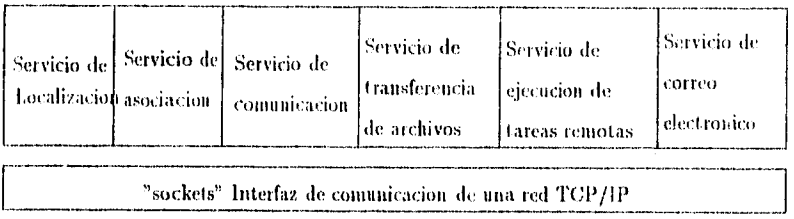


Fig. 3.8 Servicios de la Interface de Communicación

- Transmisión/Recepción de mensajes.
- Ejecución de tareas remotas.
- Transferencia de archivos.
- Correo electrónico.

De esta forma la IC puede ser apreciada en la figura 3.8.

El *servicio de localización* proporciona información acerca de los *usuarios* de la red. Dicha información contiene entre otros datos: dirección del *usuario*, su área de interés, y su identificador.
Es mediante este servicio que un *usuario* puede tener la referencia de otros *usuarios* que le puedan ayudar en un trabajo cooperativo determinado.
Las operaciones básicas que un *usuario* le pueden aplicar al *servicio de localización* son: darse de alta, baja, realizar algún cambio, y hacer alguna consulta.

El servicio de *asociación* proporciona, entre otras cosas, la facilidad de crear grupos de trabajo. Las operaciones básicas que se pueden realizar sobre estos son: su creación, su destrucción, agregar un elemento, borrar un elemento, mandar un mensaje informativo al grupo, y mandar una solicitud de algún servicio al grupo.
El servicio de *comunicación* permite el envío y la recepción de mensajes a través de la *red*. Los últimos tres servicios son autoexplicativos.

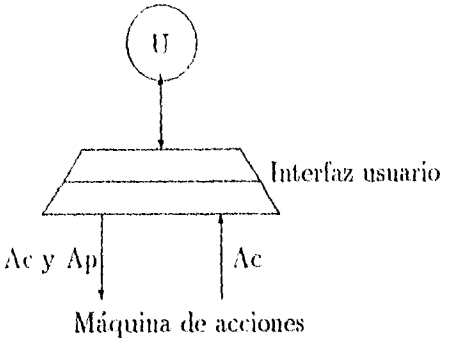


Fig. 3.9 Interfaz del Usuario

3.3.5 La Interfaz de usuario

La *interfaz de usuario* consta de dos niveles: el primero proporciona al *usuario acciones comunicativas* en la presentación específica que es manejada por éste.
El segundo nivel es la interfaz de servicios de emisión y recepción de *acciones comunicativas* y de solicitud de *acciones privadas.*
Lo anterior se ilustra en la figura 3.9.

Capítulo 4

Diseño de la IC y del SAP

4.1 Introducción

En esta sección se presenta el diseño de la *Interfaz de Comunicación* (*IC*) y del *Servidor de Acciones Privadas* (*SAP*), mismos que han sido desarrollados en el presente trabajo.

Los sistemas cooperativos forman parte del campo de la Inteligencia Artificial Distribuida (IAD).
A partir de lo expuesto en los capítulos uno y dos podemos observar que un sistema cooperativo requiere de dos elementos básicos:

1. Estrategias (o Mecanismos) de cooperación.

2. Un sistema distribuido (SD).

El primero permite definir el proceso a seguir para lograr que un grupo de agentes trabaje en forma cooperativa, con la finalidad de alcanzar un fin común.
El segundo sirve como plataforma del primero, para soportar la comunicación entre los agentes que conforman el sistema cooperativo y que permita el acceso a los recursos de la red.

De acuerdo con el análisis de las secciones de sistemas cooperativos y de sistemas distribuidos, presentados en los capítulos 1 y 2 respectivamente, consideramos que un sistema cooperativo requiere de las habilidades para:

- formar grupos de trabajo
- comunicarse
- asignar tareas
- localizar entes en la red
- utilización de los recursos proporcionados por la red

El trabajo cooperativo puede auxiliarse de la formación de grupos de agentes, de manera negociada y dinámica, lo cual sólo es posible mediante la identificación y localización de los elementos que los conforman.
La comunicación entre ellos es indispensable para que expresen sus necesidades particulares y puedan llegar a la definición de un objetivo común. Para alcanzar la realización de éste se requiere de la asignación de tareas a los agentes que pertenezcan al grupo de trabajo.
Esto es, para la realización de un trabajo cooperativo se requiere de la comunicación y de la asignación de tareas entre los agentes que conforman el grupo de trabajo. La ejecución de una tarea puede requerir del uso de los diversos recursos distribuidos que proporciona la red.
A continuación se presenta un análisis más detallado de lo anterior.

4.1.1 Asignación de tareas

Dentro de los problemas de un sistema cooperativo se encuentra la distribución de tareas. Cada una de éstas forma parte de una tarea global.
El proceso de asignación de tareas se realiza de la siguiente manera:

1. Comunicación de las necesidades particulares.
2. Negociación.
3. Obtener un objetivo común.
4. Elaboración de un plan.

En primera instancia, cada uno de los agentes le comunica a los demás sus necesidades particulares. A continuación se establece un proceso de negociación, en el cual se intercambian propuestas de opciones para lograr sus demandas. Esto conlleva a la formación de un objetivo común, a partir del cual se elabora un plan, y con éste se determina la asignación de tareas.
Al terminar cada agente con la tarea que le fue asignada, contesta con el resultado de la misma.
La ejecución de programas y de procedimientos remotos es una manera de distribuir el trabajo, similar al proceso de asignación de tareas. Sólo que en el primero no existe un proceso de negociación, no obstante es una herramienta útil para la realización del trabajo cooperativo.
El *servicio de ejecución de tareas* remotas, provee la posibilidad de ejecutar un programa remoto completo. Un ejemplo de este tipo de servicio son las RPC's que permiten la ejecución de procedimientos remotos.

4.1.2 Comunicación

En el proceso mencionado, la comunicación es un elemento fundamental. Es imposible que exista cooperación, entre un grupo de agentes, si no se encuentra presente ésta.
En un sistema cooperativo lo que se requiere comunicar básicamente es: las necesidades particulares; propuestas y contrapropuestas, en el proceso de negociación; las secuencias de actividades con las que cuenta un plan; y los resultados de la ejecución de una tarea.

Además de esto se utiliza la comunicación para informar del estado de un agente o de cualquier otra cosa que pueda ser de utilidad para los demás.

La comunicación basada en mensajes es la manera más natural y adecuada para que los agentes de un sistema cooperativo se comuniquen [LB92].

4.1.3 Grupos

Los grupos de trabajo son una herramienta que puede ser utilizada por los sistemas cooperativos.
La necesidad de formar grupos para realizar un trabajo cooperativo está fundamentado en la teoría de la racionalidad limitada [S.80], la cual expresa que la capacidad de la mente humana está limitada para resolver problemas del mundo real.
Lo mismo puede decirse de una computadora, ya que tiene un límite en la velocidad de procesamiento y en su capacidad para almacenar datos.

Un sistema cooperativo requiere organizarse mediante grupos de trabajo para superar a través de éstos las barreras impuestas por la "racionalidad limitada".

Un ejemplo de un servicio elemental para la formación de grupos es el propuesto por la extención del kernel V de Unix para la formación de grupos de procesos [CZ85], así como las listas de correo en Unix son un ejemplo elemental del servicio descrito en esta sección.

4.1.4 Recursos

Una red de comunicaciones permite, a los elementos que la conforman, acceder a los recursos con la que ésta cuenta. Es decir, que la red permite que los recursos sean compartidos por los elementos que se encuentran conectados a ella, y de esta forma se evita tener redundancia de recursos.

Los recursos pueden ser muy variados y diversos, y se pueden clasificar en privados y públicos . Los primeros son aquellos a los cuales sólo lo pueden accesar un grupo de elementos o incluso un sólo elemento.
Todos los elementos de una red pueden accesar a un recurso público.
Los recursos pueden ser de hardware, de software, y de documentos.
En el primero tenemos el caso de las impresoras, bases de datos, lectoras de cintas, etc.
En el segundo tenemos, por ejemplo, programas que ofrecen algún servicio. Y en el último caso tenemos información que puede estar contenida en varios archivos.

Otro ejemplo de recursos compartidos es el NFS de Unix, el cual permite compartir archivos (o disco) entre todos los nodos de una red.

4.1.5 Localización

Para la creación de grupos se requiere de la ubicación de los elementos que se desean agregar a éste. Para esto se necesita de un *servicio de localización*. Pero no solamente se requiere de la identificación de agentes (o elemetos), sino también de los recursos presentes en una red de comunicaciones. Como por ejemplo: una base de datos, una impresora, una cinta de almacenamiento, etc.

Es por eso que dicho *servicio de localización* puede ser pensado como el que permite la identificación de cualquier entidad que esté en la red, incluso un grupo.

El *servicio de localización* permite, que al ubicar una entidad en la red, se lleve a cabo un intercambio de información entre las entidades que conforman la red. Esto es, el directorio es un elemento que permite el inicio de un proceso comunicativo.
De la misma manera, el uso de grupos tiene la finalidad de establecer comunicación entre determinados agentes de la red.

En resumidas cuentas, tanto el *servicio de localización* como el de *asociación* son elementos de apoyo a la comunicación de un sistema cooperativo.

4.1.6 Conclusión

En resumen podemos decir que los servicios deseables que debe proporcionar la plataforma del SD que soporte a un sistema cooperativo son:

- servicio de localización
- servicio de asociación

- servicio de comunicación
- servicio de transferencia de archivos
- servicio de ejecución de tareas remotas
- servicio de correo electrónico

En un sistema cooperativo el *servicio de localización* nos permite la ubicación de los agentes con los cuales se desea trabajar, estos pueden ser agrupados mediante el *servicio de asociación*. Y a través del *servicio de comunicación* pueden interaccionar unos con otros.
Los servicios de *transferencia de archivos y correo electrónico* también son útiles para el intercambio de información entre los agentes.
El *servicio de ejecución de tareas remotas* es una herramienta que puede ser útil para la ejecución distribuida de tareas.

Como puede observarse las funciones que soportan el trabajo cooperativo son variadas y pueden ser simples o complejas.
Las primeras serán proporcionadas por la IC y las segundas por el SAP. Sin embargo, las dos en conjunto tienen la finalidad de proporcionar las herramientas necesarias para soportar el trabajo cooperativo.

Los servicios señalados arriba serán proporcionados por la IC a la *máquina de acciones privadas*, y el conjunto de ellos tiene como finalidad proporcionar los servicios básicos e indispensables para soportar la comunicación entre *facilitadores*.

A partir de estos servicios se pueden construir otros más elaborados, que faciliten y automaticen el uso de los primeros. Éstos se encuentran en el SAP. Los servicios (o *acciones privadas*) realizadas por este servidor pueden dividirse en dos:

- generales
- particulares

Las primeras son aquellas acciones privadas que pueden ser requeridas por cualquier *usuario*.

Las segundas son aquellas *acciones privadas* que son requeridas sólo por su *usuario* asociado para cubrir sus necesidades particulares.

Algunos de los servicios generales que proporcionará el SAP son:

- Transferencia de documentos. En donde un documento está integrado por un conjunto de archivos.
- Acceso a bases de datos remotas o locales.
- Ejecución de tareas remotas o locales. En donde una tarea está formado por un conjunto de programas.

La transferencia de documentos consiste en la utilización repetida del *servicio de transferencia de archivos* de la IC, hasta haber transmitido todos los archivos que conforman el documento.
De manera similar la ejecución de tareas puede estar formando por la utilización repetida de varios servicios de la IC o del SAP.

Por ejemplo una tarea puede estar construida por:

- la ejecución de varios programas
- el acceso a una base de datos
- la transferencia de un documento

La filosofía que sigue el diseño de la IC y del SAP es aprovechar e integrar los servicios ya existentes de la red TCP/IP-Unix. En este caso existen cuatro servicios de los propuestos para la IC:

- servicio de comunicación
- servicio de transferencia de archivos
- servicio de ejecución de programas remotos
- servicio de correo electrónico

4.2 Diseño de la Interfaz de Comunicación

Como ya se mencionó antes, la *IC* tiene la finalidad de proporcionar, al SAP, ciertos servicios básicos informáticos y de permitir la recepción y transmisión de *acciones comunicativas* entre la *máquina de acciones* y la *red*.

Esta interfaz debe proporcionar los servicios indispensables para que los *facilitadores* puedan comunicarse, a través de la red, para realizar un trabajo cooperativo.
Dicho trabajo implica la colaboración de un grupo de *usuarios*, de la misma manera que la cooperación entre seres humanos sólo puede darse mediante grupos de trabajo.
Como ya se indicó anteriormente, un *usuario* puede ser un elemento humano o informático.

El proceso de formación de un grupo requiere de un organizador que tiene la habilidad para contactar a los elementos que quiere invitar para formar su grupo.
Esta habilidad puede pensarse como un *servicio de localización* mediante el cual se pueda buscar un *usuario*.

4.2.1 Servicio de Localización

El *servicio de localización* constará del acceso a varios directorios, de los cuales se hablará más adelante.
El formato básico de un directorio está integrado por una serie de patrones.
Cada patrón está formado por un identificador único, un conjunto de características parciales y la dirección de la entidad que representa dicho patrón.
Es decir:
identificador único;caraterísticas parciales;dirección

Como ya se mencionó, la localización puede ser llevado a cabo mediante:

- un identificador único

- características parciales

Cada uno de estos tiene una manera de operar diferente, como se mostrará a continuación.
Cada uno de los elementos contenidos en un directorio puede estar en tres tipos de estados:

- activo
- pasivo
- ocupado

En el primero el *usuario* se encuentra presente. En el segundo, no se encuentra activo, y por lo tanto no podrá atender ningún llamado. Y en el tercero se encuentra demasiado ocupado para atender a alguien.

Localización por identificador único
La localización por identificador único requiere que el patrón de búsqueda tenga cuando menos definido dicho identificador, por ejemplo:
nombre único: Juan
característica$_1$*: —*
característica$_2$*: —*
característica$_3$*: —*
. . . dirección: —

Localización por características parciales
La localización por características parciales significa que el patrón de búsqueda carece de el identificador único, más sin embargo, cuenta con alguna(s) características del ente buscado, por ejemplo:
nombre único: —
característica$_1$*: —*
característica$_2$*: cualidad*$_x$
característica$_3$*: —*
. . . dirección: —

En los dos casos de búsqueda el *servicio de localización* proporcionará el patrón asociado al ente buscado. En caso de que sólo se

necesite la dirección, ésta podrá ser extraída del patrón, y está formada por:

- una dirección IP
- un número de puerto

El *servicio de localización* ofrece apoyo para la formación de grupos de trabajo, en los cuales los *usuarios* integrantes de estos cumplen con un patrón determinado.
Por ejemplo, si se deseara formar un grupo de *usuarios* con capacidades en Inteligencia Artificial Distribuida (IAD), y en Sistemas Expertos (SE) con temas de interés en: bases de datos distribuidas (BDD) y en sistemas orientados a objetos (SOO). El patrón que se debería utilizar está formado, básicamente, de la siguiente manera:
Nombre: —
Estado: activo
Temas de interés: BDD,SOO
Capacidades: IAD,SE
Dirección: —

El *servicio de localización* también permite obtener información acerca de un ente integrado a la red. Incluso puede ser que se conozca su dirección pero se ignoren ciertas características que en un momento dado se requieran poder acceder a ellas.
Por ejemplo, se conoce que el *usuario* Juan tiene la dirección: dir IP=132.248.204.1, Puerto=5000. Sin embargo, se desconocen sus temas de interés y sus capacidades. El patrón de búsqueda sería:
Nombre: Juan
Estado: —
Temas de interés: —
Capacidades: —
Dirección: —

Y la respuesta que ofrecería el *servicio de localización* sería el patrón:
Nombre: Juan

Estado: activo
Temas de interés: BDD
Capacidades: IAD
Dirección: 132.248.204.1.5000

Es importante mencionar que los servicios de formar grupos de *usuarios* que cumplan con cierto patrón y el de mandar ejecutar una tarea por el *usuario* que mejor cumpla con un patrón, no son proporcionados por el *servicio de localización* [1], sino que éste permite y facilita la elaboración de estos.

Es claro que el *servicio de localización* no sólo puede ser aplicado a *usuarios*, ya que también es de gran utilidad para tener acceso a un directorio de grupos, así como de recursos de la red como impresoras, cintas de almacenamiento, bases de datos, etc.
En el caso del directorio de grupos puede darse el caso de que un *usuario* trabaje muy frecuentemente con un grupo determinado, entonces, mediante este directorio puede "activarse", cada vez que se necesite, de una manera más sencilla y automática.

En el caso de los recursos de red, tenemos por ejemplo el realizar una consulta en una base de datos, la cual no sabemos dónde está ubicada, sobre los clientes bancarios de un determinado banco y de una determinada sucursal. Para encontrar la localización de ésta se utilizaría el directorio de recursos de red, realizando una búsqueda con un patrón formado por el tipo de recurso, en este caso una base de datos, el nombre del banco y la clave de la sucursal.

Formato de directorios
El *servicio de localización* tendrá acceso a los siguientes directorios, que en conjunto conforman el *directorio general*:

- directorio de *usuarios*
- directorio de grupos sin jerarquía

[1] Son proporcionados por el *servidor de acciones privadas*, como se verá más adelante.

- directorio de grupos jerárquicos
- directorio de recursos de red

Los patrones contenidos en el directorio de *usuarios* y el de grupos (jerárquicos o no jerárquicos) estará integrado de la siguiente manera: *nombre, clase, estado, línea de función o producción, área de trabajo, tema de interés, capacidades, grupos a los que pertenece, dirección.*

El campo de clase se refiere al tipo de entidad al cual se refiere el patrón, y puede ser:

- usuario
  - humano
  - aplicación
- grupo sin jerarquía
- grupo jerárquico
  - simple
  - uniforme
  - dividido por función o por producto
- recurso de red

La clasificación anterior esta basada en la propuesta por Fox [S.80]. Los patrones contenidos en el directorio de recursos de red que estarán integrados de la siguiente manera:
nombre, clase, tipo recurso, estado, capacidades, área de trabajo, dirección.
Un ejemplo para este caso es el siguiente patrón:
imp-est:recurso:impresora-laser:activo:imprimir,color:estadística, geografía,132.248.51.21,7001
Este patrón nos indica que el nombre del recurso es *imp-est*, el cual se encuentra activo. Este recurso es una impresora a color y es ocupada

por el departamento de estadística y geografía, y su dirección consta de la dirección IP 132.248.51.21 asociada al puerto 7001.

Operaciones sobre directorios

Se ha tomado como antecedentes, para el desarrollo de las operaciones sobre directorios, a la tesis de licenciatura [Sal] desarrollada en las instalaciones de LANIA.

Las operaciones que se pueden realizar sobre un directorio son:

- creación de un directorio privado
- alta pública o privada de una entidad
- actualización del directorio público
- baja temporal de una entidad en un directorio público o privado
- nueva alta de una entidad en un directorio público o privado
- baja definitiva pública o privada de una entidad
- cambio público o privado
- consulta pública o privada

Donde una entidad puede ser:

- un *usuario*
- un grupo sin jerarquía
- un grupo jerárquico
- un recurso de red

Y un *usuario* puede ser:

- un elemento humano
- un sistema experto o una aplicación

La *creación de un directorio privado* sólo crea un directorio local, es decir, que éste no se encontrará repetido para cada *facilitador*. Esta operación puede ser realizada por cualquier *usuario*.

Una *alta* permite incluir una entidad en el directorio general. Mediante la *actualización* un *usuario* puede obtener la información contenida en el directorio público y registrarla en su directorio.
La *baja temporal* borra la información lógicamente, más no físicamente, del directorio, aunque aparenta ya no existir. De manera que si se hace una consulta buscando una entidad en este caso, la operación devolverá como resultado la no existencia de la información buscada.
Una *nueva alta* permite hacer visible nuevamente la entidad que fue dada de baja temporalmente.
La *baja definitiva* borra información físicamete, lo cual significa que es irrecuperable.
La operación *cambio* permitirá actualizar la información del directorio privado o público, realizando modificaciones de un patrón y/o de su dirección asociada.
La operación *consulta* permite hacer búsquedas de entidades mediante el uso de patrones, como ya se ha mencionado anteriormente.

Criterios de implementación

La información de los directorios puede estar ubicada en forma centralizada o distribuida. En el caso de ser centralizada, la información está en un sólo lugar, y todos los que la solicitaran la buscarían en ese lugar. Esta opción tiene la desventaja de no ser lo suficiente confiable, ya que si por alguna razón se cae el nodo donde se encuentra la información, el sistema de *servicio de localización* fallaría. En el caso distribuido, los directorios se encuentran repetidos para cada *facilitador*. Éste es un sistema tolerable a fallas, aunque la desventaja es que hay redundancia en la información.
Sin embargo, para el diseño de este servicio se ha decidido utilizar el modelo distribuido con el objetivo de obtener una mayor confiabiabilidad en el sistema.
A pesar de esto, se prevee la posibilidad de tener un directorio público y otro privado para los grupos. Con el público todos los *facilitadores* tienen una copia de éste y por lo tanto tiene libre acceso a la infor-

mación.
En el caso del privado, solamente el dueño de éste tiene la información. Es decir, que un directorio privado no se encuentra repetido para cada *facilitador*, sino que sólo existe uno al cual nada más tiene acceso el dueño.

El directorio privado de grupos tiene sentido para aquellos grupos para los cuales sólo son usados por un *usuario*. Siendo así, no tiene caso tener repetida la información que nunca será usada por los demás *usuarios*.

Una de las características deseables en el trabajo de grupo es que los *usuarios*, grupos, y recursos de red puedan cambiar su ubicación sin afectar y de manera totalmente transparente al sistema de la *red de facilitadores*.
Incluso que puedan ser agregados nuevos elementos a la red sin que éstos causen estragos en la misma.
Las facilidades que proporcionará el *servicio de localización* son similares a los que la *red de contratos* ofrece, en el sentido de que la red es completamente dinámica. Es decir, se prodrá agrandar o disminuir, e incluso mover de lugar los nodos, sin que esto afecte el correcto funcionamiento del sistema.

4.2.2 Servicio de Asociación

Como ya se mencionó, el trabajo cooperativo puede utilizar la formación de grupos, los cuales pueden variar en la complejidad de su organización, tal como lo expone Fox [S.80] y Mallone [Tho86].

Gasser ha planteado la necesidad de un lenguaje concurrente basado en objetos, como plataforma para la IAD, que soporte la formación de organizaciones [LB92].
Aunque las organizaciones que se puedan construir en dicha plataforma sean muy simples, proporcionan una gran ayuda a los sistemas de IAD para construir sobre éstas otras que sean mucho más complejas.
Por esta razón se ha decidido incluir en el *servicio de asociación* la capacidad de crear:

- grupos sin jerarquía
- grupos jerárquicos
  - simples
  - uniformes
  - divididos por función o por producto

Lo anterior, basado en lo propuesto por Fox [S.80].
Donde los elementos que conforman a los primeros no tiene ninguna jerarquía, mientras que en los segundos existe una jerarquía simple, una multinivel, y una dividida por función o por producto respectivamente.
Los grupos jerárquicos contarán con una recepción y con un directorio privado, como se indica más adelante.
En todos los grupos los elementos que los conformen podrán ser de diversos tipos.

Tipos de grupos

Como se menciona arriba en la sección de grupos, el primero puede ser utilizado para grupos no muy grandes, sin embargo, cuando estos crecen considerablemente es mejor utilizar un grupo con jerarquía simple, el cual consta de dos niveles, en el primero se encuentra un tomador de decisiones que coordina a todos los demás. Esto es debido a que el costo de la comunicación y de la distribución de la información se incrementa considerablemente.

Al crecer la dimensión del único elemento de producción de la jerarquía simple, el tomador de decisiones es incapaz de procesar toda la información y entonces se requiere de una jerarquía uniforme. Ésta tiene niveles múltiples, en donde cada uno de estos funcionan como un filtro para su nivel superior.
Al incrementarse el tamaño de esta última organización así como el número de sus productos, ocurre una competencia de recursos y de esta manera surgen grupos jerárquicos cuyas divisiones son en base a líneas de producción o de funciones.
La organización de estos dos tipos de grupos se encuentra en un mismo nivel de organización, según Fox [S.80], sin embargo, cada uno de estos se adecúan mejor a un tipo de problema diferente uno del otro. Por

ejemplo, es más adecuada la organización por función de un sistema operativo de propósito general. Esto es, dividido por un sistema de archivos, un sistema de entrada/salida, un administrador de procesos, etc. Por otro lado, si se requiere un sitema operativo para programar solamente en Basic y en Pascal, sería entonces mejor, por cuestiones de eficiencia, dividir al sistema por producto. Es decir, se tendría un sistema de archivos, un sistema de entrada/salida, etc; para cada tipo de programación.

Cuando crece la producción y el tamaño de este tipo de organizaciones, también crece el problema de coordinación y la información a ser procesada. Entonces se requiere de una organización de mercado, en la cual no se necesita crear una nueva unidad por cada función o producto adicional que se desee, sino que puede ser contratada en el mercado. La *red de contratos* sigue este mismo esquema. Incluso se puede utilizar este protocolo para construir, a partir de un grupo dividido por función o por producto, una organización de mercado. De manera que cuando se necesite una función o un producto, se lance un mensaje anunciando dicha demanda. Al cual contestaran algunos, *usuarios* y/o grupos, con una oferta. De los cuales eligirá al abastecedor con la oferta más atractiva, y lo incluirá en su grupo.

Se ha decidido no crear un grupo para organizaciones de mercado, por la razón de que éstas pueden ser creadas a partir de grupos divididos por función o por producto, en donde los elementos subsecuentes que se les agreguen sean a través del protocolo de la *red de contratos* o alguno similar.

El patrón evolutivo organizacional del *servicio de asociación* queda definido como se muestra en la figura 4.1.
Todos ellos son una organización con diferente nivel de complejidad.

Tipos de elementos para cada grupo
Un elemento de un grupo sin jerarquía puede ser:

- un *usuario*
- un grupo sin jerarquía

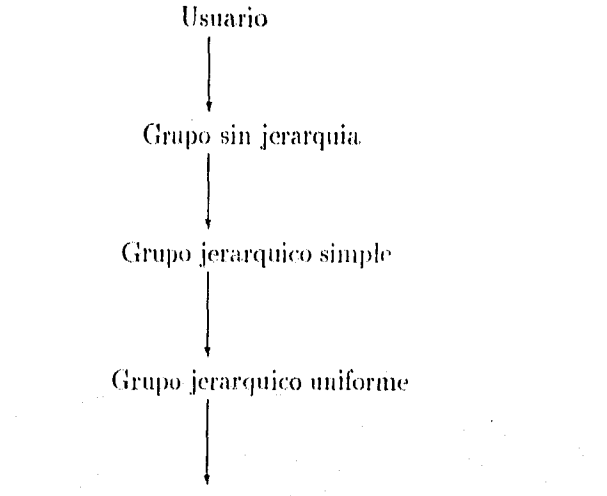


Fig. 4.1 Patrón evolutivo de las Organizaciones

Un elemento de un grupo jerárquico simple puede ser:

- un *usuario*
- un grupo sin jerarquía

Un elemento de un grupo jerárquico uniforme puede ser:

- un *usuario*
- un grupo sin jerarquía
- un grupo con jerarquía simple
- un grupo con jerarquía uniforme

Un elemento de un grupo dividido por función o por producto puede ser:

- un *usuario*
- un grupo sin jerarquía
- un grupo con jerarquía simple
- un grupo con jerarquía uniforme
- un grupo con jerarquía dividida por función o por producto

La recepción

Los grupos jerárquicos que se proponen en este trabajo retoman la idea de tener una recepción similar a la manera descrita en [HI91].
Un grupo jerárquico está formada por tres componentes:

- un conjunto de elementos
- una recepción
- un directorio privado de los elementos pertenecientes al grupo

La segunda se encarga de recibir todos los mensajes que lleguen al grupo y redireccionarlos a los elementos que correspondan dichos mensajes. Sólo la recepción tiene acceso al directorio privado.

El directorio privado

El directorio privado que requiere el grupo jerárquico simple para el manejo de sus elementos está integrado por:

- un directorio de *usuarios*
- un directorio de grupos sin jerarquía

El directorio privado que requiere el grupo jerárquico uniforme para el manejo de sus elementos está integrado por:

- un directorio de *usuarios*
- un directorio de grupos sin jerarquía
- un directorio de grupos jerárquicos simples

- un directorio de grupos jerárquicos uniformes

El directorio privado que requiere el grupo jerárquico dividido por función o por producto para el manejo de sus elementos está integrado por:

- un directorio de *usuarios*
- un directorio de grupos sin jerarquía
- un directorio de grupos jerárquicos simples
- un directorio de grupos jerárquicos uniformes
- un directorio de grupos jerárquicos divididos por función o por producto

En los tres tipos de grupos jerárquicos se utiliza el *servicio de localización* para el manejo de sus directorios privados. Y el directorio privado de un grupo jerárquico es accesado solamente por la recepción del mismo.

Operaciones sobre grupos

Las operaciones que se pueden realizar sobre grupos son:

- crear un grupo
- destruir un grupo
- agregar un elemento
- eliminar un elemento
- consultar si está un elemento determinado
- consultar el número de elementos
- solicitud
  - única
  - múltiple

- informar
- pedir otra respuesta
- almacenar en un directorio público o privado el grupo
- altas, bajas, cambios, y búsqueda de elementos en el directorio privado
- cambio de dueño

La *creación* de un grupo puede ser realizada por cualquier *usuario*. Esta operación crea un grupo sin elementos, con una recepción y su directorio privado que inicialmente no tiene información; estos dos últimos sólo en el caso de grupos jerárquicos. El creador del grupo se convierte en el dueño del mismo, y sólo este puede *destruirlo*, y con esto, borrar el directorio privado en el caso de grupos jerárquicos.

La operación de *agregar un elemento* sólo puede ser realizada por el dueño del grupo. En un grupo jerárquico simple sólo se puede agregar elementos en el segundo nivel, sin embargo, en los de jerarquía uniforme y divididas por función o por producto se tiene que dirigir la operación al nivel dónde se desea agregar el elemento.
Por ejemplo si tenemos la jerarquía uniforme x formada por los elementos:

- el *usuario* y
- un grupo jerárquico z

Si se desea agregar el *usuario* k al grupo x en un nivel dentro del elemento z, entonces la operación de *agregar* deberá dirigirse directamente al grupo z. La misma situación sucede para la *eliminación de un elemento*. Estas operaciones avisan al *usuario* que ha sido agregado o eliminado, respectivamente. Y en el caso de un grupo jerárquico, se actualiza la información de su directorio privado.

La *consulta del número de elementos* proporciona la cantidad que contiene un grupo, sin contar los de los subgrupos que éste pueda poseer.

La operación de *consultar si está un elemento* no investiga si éste se encuentra en los subgrupos que la organización pueda contener. La respuesta a esta operación puede ser negativa o positiva. Estas dos operaciones pueden ser realizadas por cualquier *usuario*.

La operación de *informar*, distribuye una copia de la información a cada uno de los elementos integrantes de un grupo sin jerarquía. En el caso de los grupos jerárquicos, la *recepción* se encarga de analizar el patrón recibido, y con ayuda de su directorio privado reenvía el mensaje a los elementos que cumplan con el patrón requerido.
Si algún elemento de los que reciben el mensaje reenviado por la *recepción* también es un grupo jerárquico, entonces realiza el mismo procedimiento hasta que finalmente el mensaje sea recibido por *usuarios*.

Por otro lado, la de *solicitud única* pide un servicio a un grupo y será atendida por un sólo elemento, elegido aleatoriamente.
El el caso de la *solicitud múltiple*, pide un servicio a los integrantes de un grupo a la cual cada uno de estos ofrece una respuesta en partipcular, las cuales son alamcenadas en una cola de respuestas. La primera de éstas es enviada al solicitante, y en caso de que el éste *pida otra respuesta*, se le proporciona la siguiente en la cola de respuestas.
Estas dos operaciones podrán ser realizadas por cualquier *usuario*.

La operación de *almacenar*, guarda toda la información del grupo en un directorio público o privado. Esta información incluye los usuarios del grupo y de los subgrupos, así como la información de los directorios privados del grupo y de cada subgrupo; ésta última en el caso de grupos jerárquicos.
El *cambio de dueño* sólo lo puede hacer el dueño del grupo.
Las altas, bajas, cambios y consultas del directorio privado, sólo pueden ser realizadas por la recepción.

4.2.3 Servicio de Comunicación

El servicio de comunicación permitirá el envío y la recepción de mensajes a través de la red.

La comunicación basada en mensajes es la manera más natural y adecuada para que los agentes de un sistema de IAD se comuniquen [LB92]. Cuando los agentes se encuentran negociando, el paso de mensajes les ofrece una herramienta de comunicación que les permite realizar la negociación de una manera interactiva y dinámica. En la cual se mandan propuestas y contrapropuestas entre los agentes cooperantes.

Esta forma de comunicación permite una interacción entre los agentes, los cuales están distribuidos y son independientes unos de otros.

4.2.4 Transferencia de archivos

La *transferencia de archivos* permitirá transportar archivos de un lugar a otro, a través de la red.
Este servicio es proporcionado por la red TCP/IP-Unix.
Las operaciones básicas que ofrece este servicio son:

- obtener un archivo
- poner un archivo

La primera permite copiar un archivo que se encuentra en un nodo remoto al nodo local. La segunda copia un archivo del nodo local a un nodo remoto.

4.2.5 Ejecución de programas remotos

Este servicio permitirá mandar realizar la ejecución de un programa que se encuentra en un nodo remoto. Es capaz de mandarle los parámetros de entrada y al terminar la ejecución el programa, regresar el resultado al nodo local. Este servcio está disponible en una red TCP/IP-Unix.

4.2.6 Correo electrónico

El servicio de *correo electrónico* permitirá mandar información en forma indirecta, esto significa, que no se requiere que el *usuario*, al cual se le envía un correo, esté presente en el momento que el correo es enviado.

Ya que éste es almacenado en un buzón, donde el usuario dueño de éste podrá tener acceso a la información que contenga en el instante que él lo desee.
Las operaciones básicas que proporcionará este servicio son:

- mandar un correo
- revisar el buzón
- leer una carta
- borrar una carta

El *mandar un correo* permite enviar información a un buzón cuya dirección es indicada.
Al *revisar el buzón* se tiene acceso a todas las cartas recibidas. *Leer una carta* permite accesar el contenido de éstas, y al *borrarla* se elimina la información del buzón.

4.3 Diseño de el SAP

Como ya se ha mencionado, el *Servidor de Acciones Privadas* (SAP) proporciona al *usuario* y a la *máquina de planes*, servicios informáticos; los cuales pueden ser locales o remotos.
El SAP se vale de los servicios de la IC para construir a partir de los servicios básicos que proporciona ésta, otros más elaborados.
Como ya se mencionó anteriormente, el SAP es el ejecutor de las *acciones privadas*, las cuales son acciones que permiten la realización de un servicio informático local o en red. Éstas son solicitadas por el *usuario* y por la *máquina de planes*.

Si la IC, en la metáfora de la oficina, equivale a los medios de comunicación con que cuenta la secretaria como son el fax, el teléfono, el servicio de mensajería, etc; para el SAP equivale a los de ciertos servicios de comunicación, pero más elaborados en el sentido de que automatizan el uso los primeros.
Pensemos por ejemplo en un fax programable, el cual tiene registrado en memoria un conjunto de personas; y la secretaria puede mandar un

fax a todas las personas, de las que tiene registradas, que cumplan con ciertas características que le son indicadas por ésta a dicho fax.

4.3.1 Operaciones sobre el SAP

Básicamente el SAP está integrado por los siguietes servicios:

- búsqueda y transferencia de archivos
- transferencia de documentos
- acceso a bases de datos locales o remotas
- ejecución de tareas locales o remotas
- manda correo electrónico a los *usuarios* que coincidan con cierto patrón
- dado un *patrón usuario* crear un grupo sin jerarquía
- dado un *patrón elemento* crear un grupo jerárquico

A continuación se expone de lo que trata cada uno de estos servicios.

4.3.2 Transferencia de documentos

La transferencia de documentos es un elemento que ayuda al desempeño del trabajo cooperativo, ya que éste contribuye como herramienta de comunicación entre los *facilitadores* cooperantes.
Un documento está integrado por un conjunto de archivos, los cuales pueden representar: texto, imagen o sonido.

Algunas de las características deseables dentro de este contexto son las siguientes:

- transferencia de un documento
- transferencia de algún archivo o documento que se ignore dónde se encuentra

- transferencia de algún archivo o documento del cual sólo se conoce su nombre y fecha de edición
- transferir un archivo o documento a los *usuarios* que cumplan con cierto patrón

Los primeros dos servicios nos ofrecen transparencia en la búsqueda de los archivos.
El último permite realizar en una sola operación varias acciones: buscar en el directorio de *usuarios* y transferir el archivo a cada uno de los *usuarios* que coincidan con el patrón establecido.

4.3.3 Acceso a bases de datos

Algunas de la características deseables para el acceso de las bases de datos (locales o remotas) son:

- capacidad para realizar, de manera sencilla, operaciones sobre una base de datos, cuando menos las que sean más indispensables
- capacidad de realizar una operación sobre una base de datos, de la cual no se conoce su ubicación o identificación

La primera puede incluir por ejemplo la programación en SQL de las consultas más frecuentes que se realicen a una base de datos determinada.
La segunda se refiere a la habilidad de mandar hacer una operación en la base de datos que cumpla con cierto patrón, de manera que ésta deberá estar registrada en el directorio de recursos de red; ya que en éste, el SAP, realizará la búsqueda y al obtener la referencia de este recurso mandará ejecutar la operación solicitada.

4.3.4 Ejecución de tareas

Una tarea consiste de un conjunto de operaciones, las cuales pueden incluir por ejemplo: la ejecución de distintos programas locales y/o remotos, acceso a bases de datos locales y/o remotas, transferencia de documentos, etc. Los servicios básicos, en cuanto a ejecución de programas locales y remotos, son:

- mandar ejecutar un programa en una hora y día en específico
- mandar ejecutar un programa periódicamente a determinada hora
- capacidad para abortar la ejecución de un programa
- mandar ejecutar una tarea al *usuario* que coincida con cierto patrón

Los dos primeros son útiles en la automatización de la ejecución de programas. La capacidad de mandar abortar es a veces necesario para mantener el correcto funcionamiento de un sistema. El último consiste por ejemplo en indicarle al SAP que requiero que me realicen la multiplicación de las matrices A y B.
En este caso no se está mencionando quién es el indicado para realizar la operación. Sino que se le deja la tarea al SAP de que encuentre al *usuario* más indicado y cuando éste termine entregue el resultado al solicitante de la operación.
La búsqueda del *usuario* la realiza el SAP en el directorio de *usuarios* con el *servicio de localización* y mediante el patrón que se le haya proporcionado.

4.3.5 Correo electrónico

El envío de correo electrónico a los usuarios que coincidan con cierto patrón es una *acción privada* que posibilita que se entregue la información sólo a aquellos *usuarios* que les pueda interesar ésta. Y el proceso es simple, ya que no se requiere primero preguntarle al *usuario* si le interesa y luego sí la respuesta es positiva, entregarle la información.

4.3.6 Dado un patrón crear un grupo

La *acción privada* de crear un grupo dado un patrón, puede ser de ayuda significativa para el trabajo cooperativo.
Téngase por ejemplo que se requiere crear un grupo que esté interesado en el tema de "procesamiento digital de imágenes" con el objetivo de realizar intercambio de ideas sobre éste.

Otro ejemplo es desear crear un grupo de los *usuarios* que estén trabajando en un proyecto determinado con la finalidad de establecer una fecha para realizar una junta.

La labor de construir un grupo que puede ser un tanto larga y tediosa puede convertirse en una operación simple si se cuenta con una *acción privada* que realice este trabajo.

Siendo así el SAP se encargaría de realizar la búsqueda de usuarios o elementos, mediante patrones en el directorio correspondiente e insertarlos en el grupo que se requiera.

Cuando un grupo es creado, a través del *servicio de asociación*, el SAP se encarga de la interpretación y manejo de éste.
A continuación se presenta un ejemplo que muestra el manejo de los cuatro tipos de grupos (sin jerarquía; con jerarquía simple, uniforme y dividida por función o por producto) llevada acabo por el SAP, y que es ilustrado en la figura 4.2.

Se tiene una organización llamada LANIA la cual es un grupo jerárquico dividido por función (GJDF) y que está conformada por:

- una recepción
- un *usuario*: el director de la organización
- un grupo sin jerarquía (GSJ): servicio social
- varias grupos jerárquicos simples (GJS):
  - tesistas
  - investigadores
  - vigilancia
  - intendencia
  - maestría en inteligencia artificial (MIA)
  - administración del equipo de cómputo (AEC)
  - diplomado

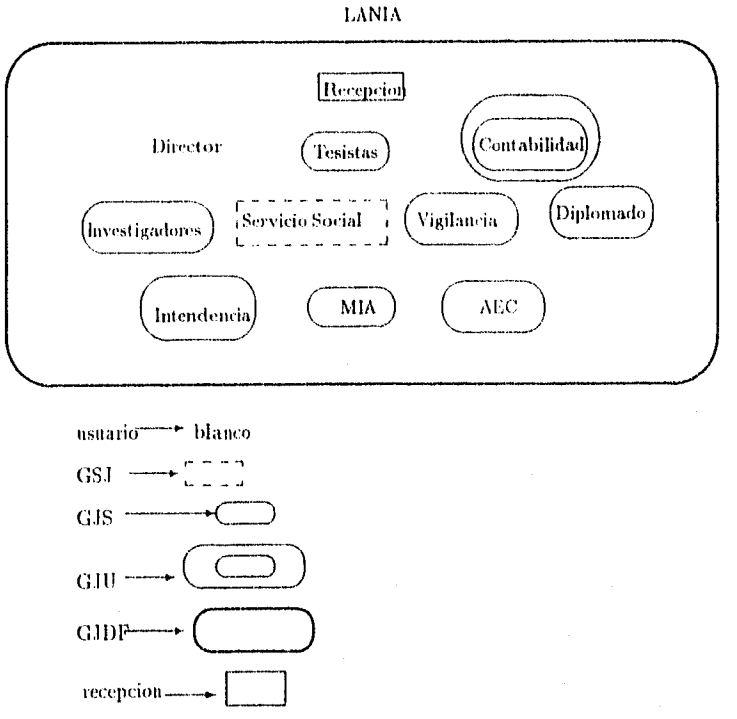


Fig. 4.2 Ejemplo de Grupos

- un grupo jerárquico uniforme (GJU):
  - contabilidad

A su vez el GJS de tesistas está formada por:

- una recepción
- un coordinador
- dos GSJ:
  - tesistas de licenciatura
  - tesistas de maestría

El GJU de contabilidad está formada por:

- una recepción
- un usuario: el contador
- un GJS: auxiliares de contabilidad

El GJS de auxiliares de contabilidad está formado por:

- un coordinador
- un GSJ de ayudantes

El GSJ de ayudantes de contabilidad está formado por:

- ayudante 1
- ayudante 2
- ayudante 3

Y así en general para todas las demás organizaciones como se muestra en la figura 4.3.

De manera que sí se quiere mandar un mensaje al coordinador del diplomando que se imparte en LANIA, solicitándole información acerca del mismo, se utiliza el siguiente patrón:
Nombre: —
Clase:usuario
Estado: activo
Función en la org: coordinador diplomando
Área de trabajo: diplomado
Temas interés: —
Capacidades: —
Grupos a los que pertenece: —
Dirección: —

Al recibir la recepción de LANIA un mensaje con este patrón, buscaría en el directorio de la organización de LANIA la dirección del diplomado y al encontrarla enrutaría hacia éste el mensaje. Al recibirlo la recepción del diplomado buscaría en su directorio la dirección del coordinador del diplomado al cual finalmente entregaría el mensaje. A continuación éste contestaría, de manera directa a quien se lo solicitó, con la información requerida del diplomado.

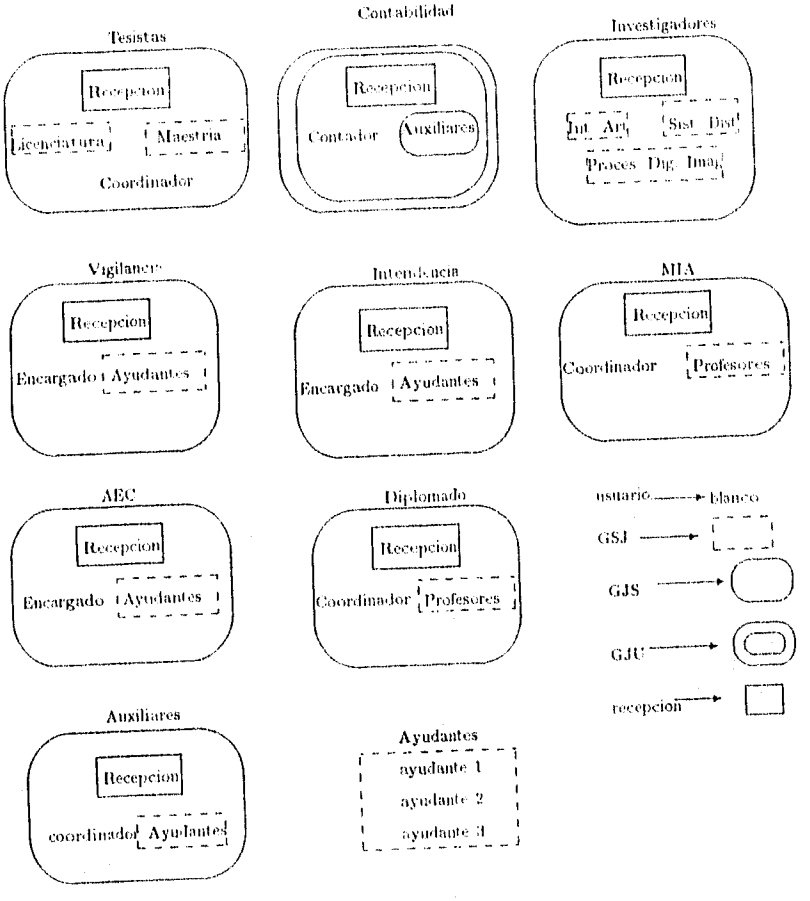


Fig. 4.3 Integrantes de los grupos contenidos en la organización de LANIA

El campo *clase* nos indica si el elemento buscado es un elemento informático, un humano, un grupo, o algún recurso de red, como una impresora.
Como lo muestra el ejemplo, esta estructura de organizaciones nos permite mandar un mensaje a una de éstas como a un todo, es decir, que no se requiere conocimiento alguno de un *usuario* o grupo que se busque en una organización, más que el patrón por el cual se identifica.

Mediante este esquema se puede mandar una solicitud de algún servicio a una organización, la cual determinará cuál es el elemento adecuado para llevarlo acabo.
También es útil para entregar información a una organización, encargandose ésta de redistribuirla a quien pueda ser de interés dentro de la misma.
Por ejemplo si se requiere mandar información acerca de un simposium en Inteligencia Artificial (IA) al personal de LANIA interesado en el tema. El patrón que se utilizaría es el siguiente:
Nombre: —
Clase: usuario
Estado: activo
Función en la org: —
Área de trabajo: investigadores, tesistas, diplomado, MIA
Temas de interés: IA
Capacidades: —
Grupos a los que pertenece: —
Dirección: —

Entonces el mensaje sería entregado a todos los investigadores, tesistas, integrantes del diplomado y de la MIA que tengan como tema de interés la IA.

Capítulo 5

Especificación de la IC

5.1 Introducción

En ésta sección se presenta la especificación del módulo de la IC, mismo que se compone de la definición del protocolo para el *servicio de asociacion* y para el de *localización* siguiendo la metodología propuesta por el modelo OSI. Esto es debido a que los demás servicios ofrecidos por la IC ya se encuentran definidos en la red TCP/IP-Unix.

En la figura 5.1 se presenta la localización lógica en el modelo OSI de los servicios proporcionados por la IC, los cuales se encuentran definidos en la capa de sesión.

7. Capa de Aplicación	*usuario*
6. Capa de Presentación	Módulo del SAP y de la MA
5. Capa de Sesión	Servicio de Localización Servicio de Asociación Servicio de Comunicación Servicio de Transferencia de archivos Servicio de Ejecución de tareas remotas Servicio de Correo electrónico
4. Capa de Transporte	Servicio de Transporte
3. Capa de Red	Red
2. Capa de Enlace	Enlace
1. Capa física	Física

Fig. 5.1 Localización lógica en el modelo OSI de los servicios de la IC

El módulo del SAP y de la MA, que se encuentran en la capa de presentación utilizan estos servicios, los cuales a su vez utilizan los ofrecidos por la capa de transporte para el envío de mensajes a través de la red. A los usuarios del *servicio de localización* se les denominará con el nombre de U_SL.
La entidad del *servicio de localización* estará denominada por el nombre E_SL.

5.2 Especificación del servicio de localización

Como ya se vió en la sección 3.4.2, las operaciones ofrecidas por el servicio de localización son las siguientes:

- creación de un directorio privado
- alta pública o privada de una entidad
- baja temporal de una entidad
- nueva alta de una entidad
- baja definitiva pública o privada de una entidad

- cambio público o privado de una entidad
- consulta pública o privada

Las primitivas básicas de comunicación que ofrece y que utiliza el *servicio de localización* para soportar las operaciones anteriores son las siguientes.

primitivas que ofrece:

Primitiva	Parametros
L_Loc.actua.solic	tipo primitiva
L_Loc.actua.conf	tipo primitiva lista de patrones
L_Loc.baja_temp_pub.solic L_Loc.alta_nueva_pub.solic L_Loc.baja_def_pub.solic L_Loc.alta_pub.solic L_Loc.consulta_pub.solic	tipo primitiva patrón
L_Loc.cambio_pub.solic	tipo primitiva patrón viejo patrón nuevo
L_Loc.consulta_pub.confirm L_Loc.consulta_priv.confirm	tipo primitiva estado de error patrón
L_Loc.alta_pub.confirm L_Loc.baja_temp_pub.confirm L_Loc.alta_nueva_pub.confirm L_Loc.baja_def_pub.confirm L_Loc.cambio_pub.confirm L_Loc.cambio_priv.confirm L_Loc.crea_priv.confirm L_Loc.baja_temp_priv.confirm	tipo primitiva estado de error

L_Loc.alta_nueva_priv.confirm
L_Loc.baja_def_priv.confirm
L_Loc.baja_temp_pub.confirm
L_Loc.alta_nueva_pub.confirm
L_Loc.baja_def_pub.confirm

L_Loc. crea_priv.solic	tipo primitiva
L_Loc.baja_temp_priv.solic	path
L_Loc.alta_nueva_priv.solic	patrón
L_Loc.baja_def_priv.solic	
L_Loc.consulta_priv.solic	
L_Loc.alta_priv.solic	

L_Loc.cambio_priv.solic	tipo primitiva
	path
	patrón viejo
	patrón nuevo

Primitivas que usa:

T_Loc.baja_temp_pub.solic	tipo primitiva
T_Loc.baja_temp_pub.ind	patrón
T_Loc.alta_nueva_pub.solic	mi dirección
T_Loc.alta_nueva_pub.ind	

T_Loc.alta_pub.solic	tipo primitiva
T_Loc.alta_pub.ind	patrón

T_Loc.cambio_pub.solic	tipo primitiva
T_Loc.cambio_pub.ind	patrón viejo
	patrón nuevo
	mi dirección

T_Loc.baja_temp_pub.resp	tipo primitiva
T_Loc.baja_temp_pub.conf	estado de error
T_Loc.alta_nueva_pub.resp	
T_Loc.alta_nueva_pub.conf	

T_Loc.baja_def_pub.resp	
T_Loc.baja_def_pub.conf	
T_Loc.cambio_pub.resp	
T_Loc.cambio_pub.conf	
T_Loc.actua.solic	tipo primitiva
T_Loc.actua.ind	tipo primitiva
T_Loc.actua.resp	mi dirección
T_Loc.actua.conf	
T_Loc.dame_info.solic	
T_Loc.dame_info.ind	
T_Loc.dame_info.resp	tipo primitiva
T_Loc.dame_info.conf	lista patrones

Las primitivas que ofrece son las que utiliza el SAP o la MAP para comunicarle al *servidor de localización* los movimientos que desea realizar sobre el directorio público o sobre alguno privado.
Las primitivas que utiliza son usadas para transmitir mensajes a través de la red y de esta manera realizar movimientos en el directorio público, el cual se encuentra distribuido. A continuación se presenta la operación y especificación del protocolo del *servicio de localización* SL.

5.2.1 Operación del protocolo del SL

Las operaciones que se pueden realizar a través del *servicio de localización* pueden dividirse en cuatro clases, de acuerdo con el protocolo que utilizan éstas.

- movimientos en el directorio público
- movimientos en los directorios privados
- alta en el directorio público
- actualización de la información del directorio público

Movimientos en el directorio público

Las siguientes operaciones tienen como finalidad realizar movimientos en el directorio público que se encuentra distribuido en la red. Este grupo de operaciones será llamado *op1* y son las siguientes.

op1:

baja_temp_pub	baja temporal en un directorio publico
baja_def_pub	baja definitiva en un directorio publico
cambio_pub	cambio en un direcctorio publico
alta_nueva_pub	alta nueva en un directorio publico

El protocolo seguido por este grupo de operaciones es el siguiente y es ilustrado en la figura 5.2.

El U_SL local utiliza la primitiva L_Loc.op1.solic para solicitarle a la E_SL local algún tipo de operación del grupo de *op1*.

En el caso de las bajas y de la alta, el U_SL envía como parámetros el tipo de primitiva y el patrón que se quiere dar de alta o de baja. En el caso de que se requiera realizar la operación de cambio se envía como parámetros al tipo de primitiva, al patrón viejo y al patrón nuevo. El segundo es el patrón que se desea modificar, y el tercero es el que tiene las modificaciones y que deberá ser reemplazado por el original.

La E_SL solicita a la capa de transporte con la primitiva T_Loc.op1.solic que comunique a todas las demás E_SL la operción que se desea realizar, para esto reciben la primitiva T_Loc.op1.ind.

Los parámetros de éstas dos últimas, en el caso de las bajas y de la alta son: tipo de primitiva, patrón, y la dirección de la E_SL local. Y para la operación de cambio son: patrón viejo, patrón nuevo y la dirección de la E_SL local.

En ambos casos el último parámetro es necesario para que las E_SL remotos tengan la referencia del solicitante, y así le puedan contestar.

De esta manera las E_SL remotas confirman la realización de la operación solicitada mediante L_Loc.op1.resp, la cual al viajar a través de la red se convierte en T_Loc.op1.conf. Los parámetros de estas primitivas para las cuatro operaciones son: el tipo de primitiva y el estado de error. A través del estado de error se puede saber si la operación fue exitosa u ocurrió alguna dificultad.

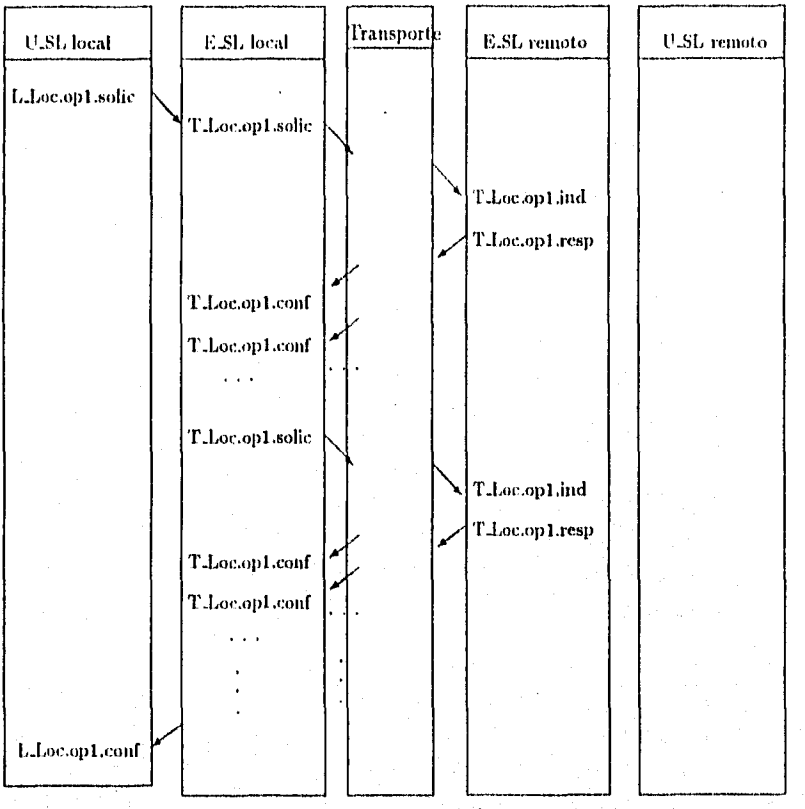


Fig. 5.2 Diagrama de secuencia temporal de primitivas para op1

Si se ha vencido un *timeout* y el número de confirmaciones recibidas no coincide con el número de E_SL remotas, la E_SL vuelve a mandar T_Loc.op1.solic, sucediendo a esta el mismo desarrollo descrito.

En el momento en el que se venza nuevamente el *timeout* y que el número de estos sea igual al número máximo de *timeout* permitidos o que se hayan recibido número igual de confirmaciones al de E_SL remotas, la E_SL local envía L_Loc.op1.conf al U_SL local, que en los cuatro operaciones cuenta con los parámetros: tipo de primitiva, y estado de error. Y mediante éste útimo se entera el U_SL si la operación solicitada fue exitosa.

Movimientos en los directorios privados

Existe otro grupo de operaciones, el cual sólo realiza acceso a los directorios locales, con excepción de una.

op2:

crea_priv	crea directorio privado
alta_priv	realiza una alta en un directorio privado
baja_temp_priv	realiza una baja temporal en un directorio privado
alta_nueva_priv	realiza una alta nueva en un directorio privado
baja_def_priv	realiza una baja definitiva en un directorio privado
cambio_priv	realiza un cambio en un directorio privado
consulta_priv	realiza una consulta en un directorio privado
consulta_pub	realiza una counsulta en un dierectorio publico

El protocolo que siguen éste grupo de operaciones es muy sencillo y es ilustrado en la figura 5.3.

El U_SL local envía la primitiva L_Loc.op2.solic para la cual la E_SL local le contesta con L_Loc.op2.conf. Para todas las operaciones, excepto la de *cambio_priv*, el U_SL manda, en la primitiva mencionada arriba, los parámetros: tipo de primitiva, path y patrón. El path indica la ruta de directorio en el cual se encuentra localizado el directorio privado del que se está haciendo referencia. El tercer parámetro indica el patrón que se requiere dar de alta, de baja, o consultar.

En el caso de que la operación sea *cambio_priv*, los parámetros son: tipo primitiva, path, patrón viejo, patrón nuevo.

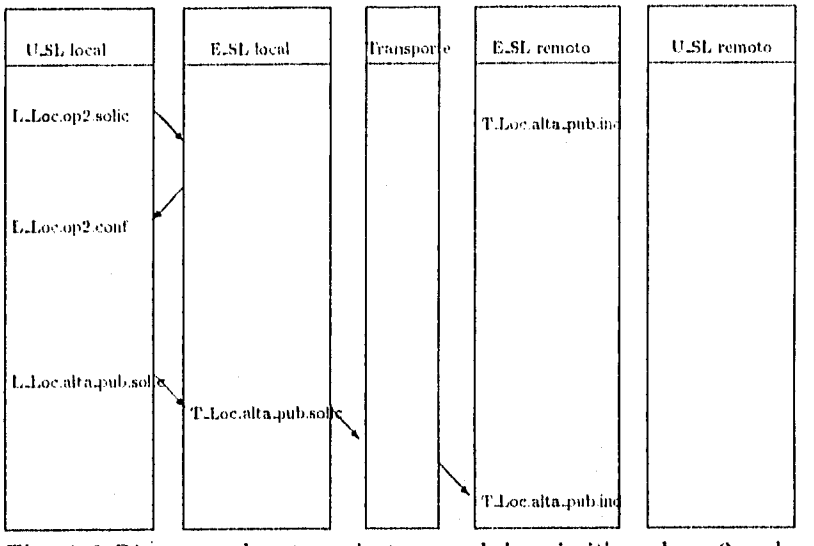


Fig. 5.3 Diagrama de secuencia temporal de primitivas de op2 y de alta_pub

El segundo parámetro indica la localización lógica del directorio privado, como en el caso anterior. El tercero es el patrón que se desea reemplazar por el último parámetro enviado.

La segunda primitiva mencionada arriba tiene asociados, para todas las operaciones, con excepción de las consultas, los parámetros: tipo de primitiva y estado de error. El segundo indica si la operción fue exitosa o sí falló.
En el caso de las operaciones de consulta, los parámetros son: tipo de primitiva, estado de error, y patrón. El segundo indica sí ocurrió alguna falla en la ejecución de la operación.
El tercero es el patrón que se obtiene como respuesta a la consulta.

Alta en el directorio público
La operación *alta_pub* ilustrada en la figura 5.3, que realiza una alta en el directorio público, sólo envía un mensaje en broadcast, indicando que se ha dado de alta un ente en la red, y de esta manera todos los E_SL

remotos pueden actualizar su información. En este caso no se requiere que sea confirmada la operación, ya que la E_SL local que ha ingresado a la red no tiene la información de cuantas E_SL remotas existen.
El protocolo que sigue ésta operación es la siguiente.
El U_SL local envía L_Loc.alta_pub.solic, a ésta la E_SL local responde enviando T_Loc.alta_pub.solic a través de la red, la cual es un mensaje en broadcast. Y por último las E_SL remotas reciben T_Loc.alta_pub.ind.

En las tres primitivas los parámetros que tienen asociados son: tipo primitiva, patrón. El segundo indica el patrón que está asociado a la entidad que se desea dar de alta.

Actualización del directorio público
La última operación, ilustrada en la figura 5.4, es la de actualización, para la cual el U_SL local envía al E_SL local L_Loc.actua.solic, cuyos parámetros asociados son únicamente el tipo de primitiva.
En seguida dicha entidad utiliza T_Loc.actua.solic, la cual envía un mensaje en brodcast a través de la red. Dicho mensaje pretende solicitar que alguna E_SL remota le indique si está dispuesto a pasarle toda la información contenida en el directorio público. Para esto, cada una de las E_SL remotas recibe T_Loc.actua.ind. Los parámetros que tiene asociados éstas dos últimas primitivas son: tipo de primitiva y su dirección. El segundo parámetro les permite a las E_SL remotas contestar de manera directa al solicitante del servicio con la primitiva T_Loc.actua.resp cuyos parámetros también son: tipo de primitiva y su dirección.
En ésta parte del protocolo, la E_SL toma la primera primitiva recibida y manda de manera directa a la dirección contenida en ésta, la primitiva T_Loc.dame_info.solic y al pasar ésta a través de la red se convierte en T_Loc.dame_info.ind, cuyos parámetros de éstas dos son: tipo de primitiva y la dirección del E_SL local. De manera que el E_SL remoto, que se ha elegido, conteste de manera directa con la información requerida utilizando T_Loc.dame_info.resp. Ésta se convierte en T_Loc.dame_info.conf al ser recibida por la E_SL local, y posteriormente éste envía al U_SL local la primitiva L_Loc.actua.conf. Los parámetros de éstas tres últimas son: tipo de primitiva y lista de patrones.

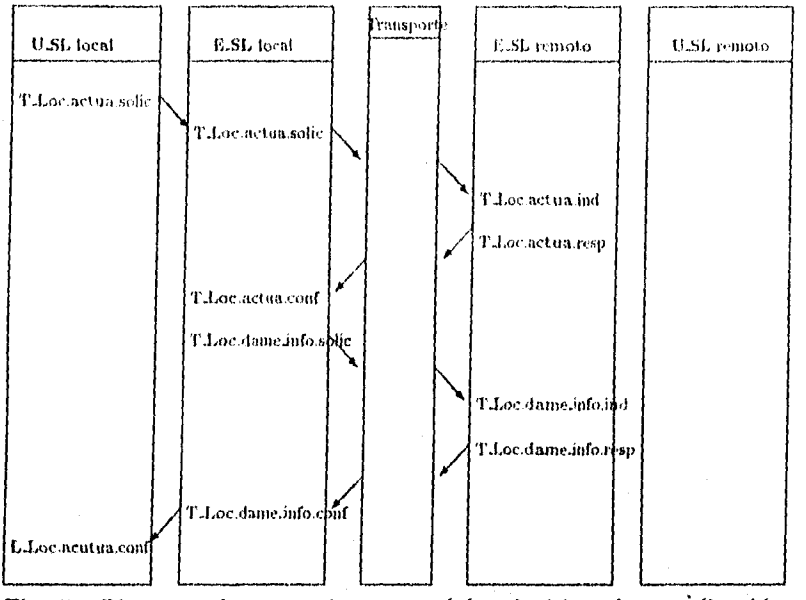


Fig. 5.4 Diagrama de secuencia temporal de primitivas de actualización del directorio público

El segundo parámetro son los datos correspondientes al directorio público.

Proceso de ingreso de un nuevo usuario

En el momento en que entra un nuevo *usuario* a la red, éste debe de anunciarse en la misma para que los demás estén enterados de su existencia, y también debe de obtener la información de todas las entidades que se encuentran en la red.

Para llevar a cabo lo anterior, el nuevo *usuario* deberá realizar la operación de *alta_pub*, la cual tan sólo informa a los E_SL remotos de la nueva entidad en la red.
Incluso cuando se agregue un nuevo recurso, como una impresora, ésta puede darse de alta con dicha operación.
No se requiere que las E_SL remotas confirmen la realización de una alta, ya que en el caso de un facilitador que entre por primera vez a

la red, éste no tendrá aún registrado las E_SL que se encuentran en la red y por tanto no tiene caso que reciba las confirmaciones de quien desconoce.

Sin embargo, cambia el protocolo para realizar una *alta_nueva_pub*, ya que en este caso sí se conoce por lo menos algunos de los *usuarios* que se encuentran en la red. Debido a que el uso de ésta operación implica que alguna vez ya se dió de alta y que por tanto conserva registrado en su directorio alguna información sobre los *usuarios* que se encuentran en la red.

En éste momento el *usuario* puede utilizar la operación de *actualizar* para obtener la información del directorio público. Siendo así , primero manda un mensaje en broadcast preguntando quien le puede pasar tal información.
A dicha mensaje contestan todas aquellas E_SL remotas que están dispuestas, pero sólo elige a la primera de la cual recibe contestación, a la cual le envía un mensaje solicitandole toda la información de su directorio público, y ésta en seguida contesta enviando lo requerido por su solicitud.

5.2.2 Especificación del protocolo del SL

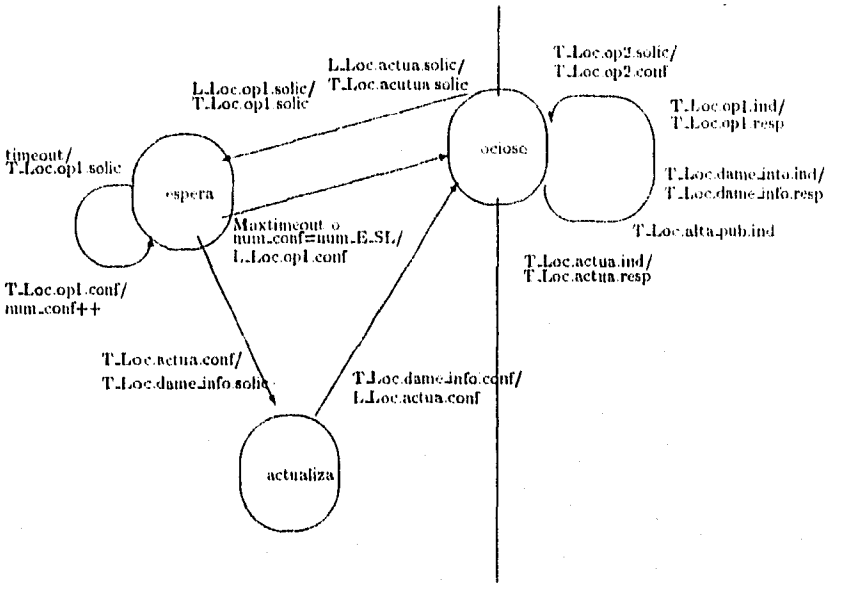


Fig. 5.5 Diagrama de estados de la E_SL

5.3 Especificación del servicio de Asociación

De las operaciones ofrecidas por el *servicio de Asociación*, mostradas en la sección 3.4.2, sólo se especificarán las que se implementarón y son las siguientes:

- crear un grupo sin jerarquía
- destruir un grupo sin jerarquía
- agregar un elemento
- eliminar un elemento
- consultar:
  - si está un elemento determinado

  - el número de elementos
  - grupos de los que soy dueño
  - grupos en los que participo
- habilitar el uso remoto de una función
- deshabilitar el uso remoto de una función
- informar a un grupo sin jerarquía
- informar a un agente
- obtener el primer mensaje de información recibido sin leer
- solicitar a un grupo múltiples respuestas
- solicitar a un grupo una respuesta
- solicitar a un agente
- pedir una respuesta
- terminar una solicitud

Proceso en el manejo de grupos

Primero y antes que todo se requiere crear el grupo con el cual se desea trabajar. Posteriormente se le pueden agregar los elementos que formarán parte de éste, o incluso eliminar alguno no deseado.

Una vez que el grupo está constituido se le puede mandar un mensaje de información o hacerle una solicitud. Las solicitudes requeridas podrán ser múltiples, ya que ésta operación no es bloqueante. Éstas dos últimas pueden ser aplicadas también a un agente, de manera que la interacción además de ser agente-grupo también puede ser agente-agente.

En el caso de que se mande información, ésta es guardada en un buffer hasta que el agente receptor desee leerla. Éste proceso es muy similar al que sigue el correo electrónico.

Con el fin de que un agente pueda restringir las solicitudes que atenderá, podrá habilitar o deshabilitar permisos que controlen el acceso de grupos y de otros agentes.

A continuación, el usuario del *servicio de Asociación* pedirá la respuesta de alguna solicitud, la cual le será proporcionada si ya está lista. Se podrá desechar una solicitud en el caso de que ya no esté interesado en ella.

Un agente podrá consultar los grupos a los que pertenece, y de los que es dueño así como determinar si un agente se encuentra en algún grupo. Ésta información le permitirá tener acceso a ellos.
Por último se podrá desintegrar el grupo, si así se desea.

Las primitivas básicas de comunicación que ofrece y que utiliza el *servicio de Asociación* para soportar las operaciones anteriores son las siguientes.

Primitivas que ofrece:

Primitiva	Parametros
A_Asoc.crea_gpo.solic	tipo_primitiva
A_Asoc.gpos_parti.solic	
A_Asoc.gpos_dueño.solic	
A_Asoc.dame_info.solic	
A_Asoc.habilita.solic	tipo_primitiva
A_Asoc.deshabilita.solic	id_func
	id_usuario
	num_veces
A_Asoc.quita_func.solic	tipo_primitiva
A_Asoc.pon_func.solic	id_func
	num_func
A_Asoc.pide_rta.solic	tipo_primitiva
A_Asoc.pide_rta.conf	id_solic

	p_sal
A_Asoc.dest_solic.solic	tipo_primitiva id_solic
A_Asoc.agrega.solic A_Asoc.elimina.solic A_Asoc.esta_elem.solic	tipo_primitiva id_gpo id_elem
A_Asoc.num_elem.solic A_Asoc.destruye.solic	tipo_primitiva id_gpo
A_Asoc.info_gpo.solic	tipo_primitiva id_gpo mensaje
A_Asoc.info_elem.solic	tipo_primitiva id_elem mensaje
A_Asoc.solic_elem.solic	tipo_primitiva id_elem id_func p_ent p_sal
A_Asoc.solic_elem.ind A_Asoc.solic_gpo.ind A_Asoc.solic_gpo_uno.ind	tipo_primitiva id_cliente id_func p_ent p_sal
A_Asoc.solic_gpo.solic A_Asoc.solic_gpo_uno.solic	tipo_primitiva id_gpo id_func p_ent

	p_sal
A_Asoc.crea_gpo.conf	resp
A_Asoc.gpos_parti.conf	
A_Asoc.gpos_dueño.conf	
A_Asoc.habilita.conf	
A_Asoc.deshabilita.conf	
A_Asoc.dame_info.conf	
A_Asoc.quita_func.conf	
A_Asoc.pon_func.conf	
A_Asoc.dest_solic.conf	
A_Asoc.agrega.conf	
A_Asoc.elimina.conf	
A_Asoc.esta_elem.conf	
A_Asoc.num_elem.conf	
A_Asoc.info_gpo.conf	
A_Asoc.info_elem.conf	
A_Asoc.destruye.conf	
A_Asoc.solic_elem.conf	
A_Asoc.solic_gpo.conf	
A_Asoc.solic_gpo_uno.conf	
A_Asoc.pide_rta.conf	resp
A_Asoc.solic_elem.resp	p_sal
A_Asoc.solic_gpo.resp	
A_Asoc.solic_gpo_uno.resp	

Primitivas que usa:

T_Asoc.agrega.solic	tipo_primitiva
T_Asoc.agrega.ind	id_gpo
T_Asoc.elimina.solic	
T_Asoc.elimina.ind	
T_Asoc.destruye.solic	
T_Asoc.destruye.ind	
T_Asoc.agrega.resp	tipo_primitiva

T_Asoc.agrega.conf	
T_Asoc.elimina.resp	
T_Asoc.elimina.conf	
T_Asoc.info_gpo.solic	tipo_primitiva
T_Asoc.info_gpo.ind	mensaje
T_Asoc.info_elem.solic	
T_Asoc.info_elem.ind	
T_Asoc.solic_elem.solic	tipo_primitiva
T_Asoc.solic_elem.ind	id_cliente
T_Asoc.solic_gpo.solic	id_func
T_Asoc.solic_gpo.ind	p_ent
T_Asoc.solic_gpo_uno.solic	p_sal
T_Asoc.solic_gpo_uno.ind	
T_Asoc.solic_elem.solic	resp
T_Asoc.solic_elem.resp	p_sal
T_Asoc.solic_elem.conf	
T_Asoc.solic_gpo.resp	
T_Asoc.solic_gpo.conf	
T_Asoc.solic_gpo_uno.resp	
T_Asoc.solic_gpo_uno.conf	

Las primitivas que ofrece son las que utiliza el SAP o la MAP para comunicarle al *servidor de Asociación* las operaciones que desea realizar sobre grupos.
Las primitivas que utiliza son usadas para transmitir mensajes a través de la red y de ésta manera realizar operaciones en que intervengan los diversos agentes que se encuentran distribuidos en la red de comunicaciones.
A continuación se presenta la operación y especificación del protocolo del *servicio de Asociación* (SA).
A los usuarios del *servicio de asociación* se les denominará con el nombre U_SA. La entidad del *servicio de asociación* estará denominada por el nombre E_SA.

5.3.1 Operación del protocolo del SA

Las operaciones mencionadas anteriormente pueden dividirse en cuatro clases, de acuerdo con el protocolo que utilizan éstas.

Clase 1
Las siguientes operaciones incluidas en ésta clase tienen como finalidad realizar consultas a grupos y configurar el acceso a las funciones o servicios que ofrezca un agente.
Ésta clase de operaciones será llamado *op1* y está integrado de la siguiente manera.
op1:

crea_gpo	creación de un grupo
gpos_parti	determina los grupos en los que participa
gpos_dueno	determina los grupos de los que es dueño
habilita	habilita el permiso de atender algun servicio que sea solicitado por algun agente
deshabilita	deshabilita el permiso de atender algún servicio que sea solicitado por algún agente
dame_info	proporciona el ultimo mensaje informativo recibido y que no ha sido leido
quita_func	le indica al SA que ya no ofrece una determinada funcion o servicio
pon_func	le indica al SA que ofrece una nueva función o servicio
pide_rta	pide una respuesta a una solicitud realizada
dest_solic	indica que se desechen todas las respuestas a una solicitud realizada
esta_elem	determina si un agente se encuentra incluido en algún grupo determinado
num_elem	proporciona el numero de elementos de un grupo

El protocolo seguido por éste grupo de operaciones es el siguiente y es ilustrado en la figura 5.6.

El U_SA local utiliza la primitiva *A_Asoc.op1.solic* para solicitarle a la E_SA local algún tipo de operación incluido en el grupo *op1*, a la cual la E_SA local contesta con *A_Asoc.op1.conf*.

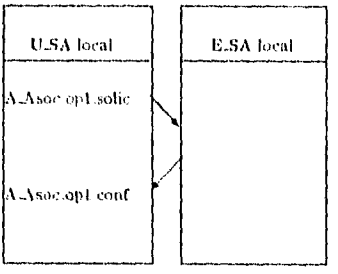


Fig. 5.6 Diagrama de secuencia temporal de primitivas de *opt*

En el caso de las operaciones *crea_gpo*, *gpos_parti*, *gpos_dueño* y *dame_info*; sólo se envía como parámetro de la primer primitiva el tipo de la misma. Para las operaciones *habilita* y *deshabilita* se envía además como parámetro *id_func*, *id_usuario* y *num_veces*. El primero se refiere al identificador de la función, el segundo al del usuario y el tercero significa el número de veces que podrá ser accesada la función por un usuario o el decremento de ellas. El usuario, en éste caso, puede ser un grupo o un agente.

Con las operaciones *quita_func* y *pon_func* se envía como parámetros a *tipo_primitiva*, *id_func* y *num_func*. El segundo es la operación que se desea agregar o eliminar, según sea el caso, y la última es su número asociado.
Los parámetros que se envían para la operación *pide_rta* son: *tipo_primitiva*, *id_solic* y *p_sal*. El segundo es el identificador de la solicitud y el tercero es el parámetro de salida que se obtiene como respuesta a una solicitud realizada.

En el caso de la operación *dest_solic* los parámetros enviados son: *tipo_prim* e *id_solic*. Al igual que en la operación anterior *id_solic* es el identificador de solicitud.
La operación *num_elem* envía como parámetros a *tipo_primitiva* e *id_gpo*. El segundo se refiere al identificador de grupo que se quiere consultar.

Por último, la operación *esta_elem* envía como parámetros a *tipo_primitiva*, *id_gpo*, e *id_elem*. El segundo es el identificador del grupo al cual se dirige la consulta, y el tercero es el del agente del cual se desea saber

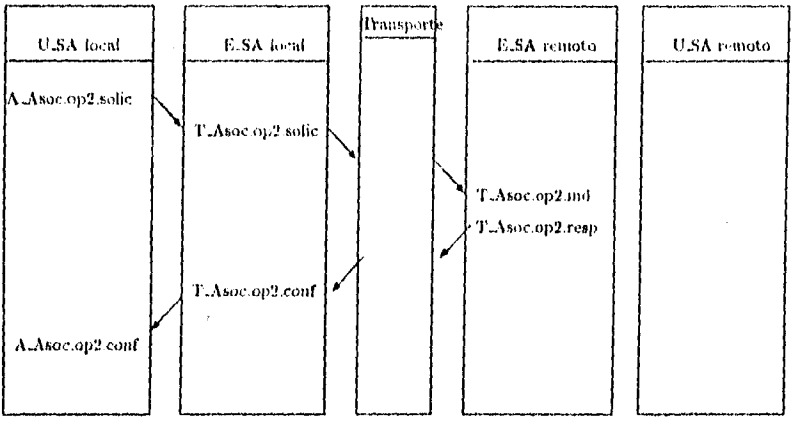


Fig. 5.7 Diagrama de secuencia temporal de primitivas de *op2*

si está presente en dicho grupo.

Todas las operaciones de grupo *op1* mandan el parámetro *resp* en la primitiva *A_Asoc.op1.conf*. En éste parámetro se pone el resultado de la operación el cual es un código de error.

Clase 2

Éstas operaciones tienen como finalidad realizar movimientos en un grupo. Dichos movimientos se refieren a agregar o eliminar elementos de un grupo.

Éste conjunto de operaciones será denominado por *op2* y está integrado de la siguiente manera.

op2:

agrega	agrega un elemento a un grupo
elimina	elimina un elemento de un grupo

El protocolo seguido por éste grupo de operaciones es el siguiente y es ilustrado en la figura 5.7.

El U_SA local envía la primitiva *A_Asoc.op2.solic* a la entidad E_SA local. Los parámetros de ésta son: *tipo_primitiva*, *id_gpo* e *id_elem*. El segundo indica el grupo al cual se requiere agregar o eliminar un ele-

mento, mismo que es identificado con *id_elem*. Al recibir ésta primitiva la E_SA local genera *T_Asoc.op2.solic*, cuyos parámetros son *tipo_primitiva* e *id_gpo*. El segundo es el identificador del agente que solicita la operación. Éste es necesario para que la E_SA remota sepa a que grupo ha sido agregado o eliminado, según sea el caso. Al pasar por la red ésta primitiva se transforma en *T_Asoc.op2.ind* y es recibida por la E_SA remota. En respuesta, ésta entidad envía *T_Asoc.op2.resp* cuyo parámetro es solamente *tipo_ptimitiva*, la cual al llegar a la E_SA local se transforma en *T_Asoc.op2.conf*.
En consecuencia ésta entidad envía al U_SA local la primitiva *A_Asoc.op2.conf* cuyo único parámetro es *resp*, el cual indicará el código de error de la operación realizada.

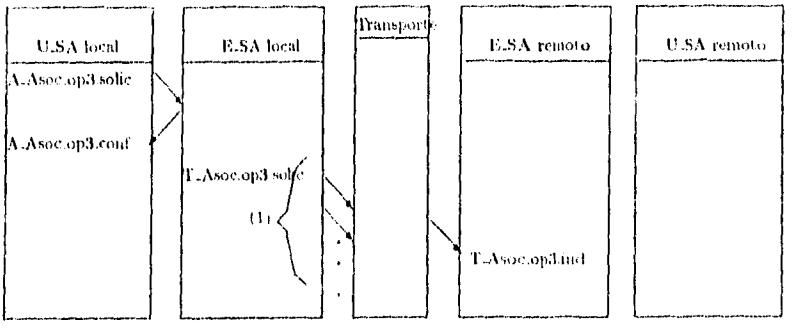
Clase 3
Las operaciones incluidas dentro de ésta clase tienen como finalidad mandar un mensaje de información a un grupo , y la de destruir un grupo.
Éste conjunto de operaciones será llamdo *op3* y está integrado de la siguiente manera.
op3:

info_gpo	manda un mensaje informativo a un grupo
destruye	destruye un grupo

El protocolo seguido por éste grupo de operaciones es el siguiente y es ilustrado en la figura 5.8.

El U_SA local envía la primitiva *A_Asoc.op3.solic* a la E_SA local. Los parámetros de ésta, para el caso de la operación destruye, son: *tipo_primitiva* e *id_gpo*. El *id_gpo* es el parámetro que identifica al grupo que se desea destruir.
Para la operación *info_gpo* los parámetros son: *tipo_primitiva, id_gpo, y mensaje*. El segundo es el identificador del grupo al cual se requiere enviar el mensaje informativo, mismo que será puesto en el tercer parámetro.

Al recibir dicha primitiva la E_SA local genera *A_Asoc.op3.conf*, y es recibida por el U_SA local. El único parámetro de ésta primitiva es



(1) n primitivas donde n=numero de elementos del grupo

Fig. 5.8 Diagrama de secuencia temporal de primitivas de *op3*

resp. En éste se le informa al usuario el código de error que ha generado la operación que solicitó.
Inmediatamente despues de enviar ésta primitiva la E_SA local genera *T_Asoc.op3.solic* y la envía a la E_SA remoto, la cual al recibirla se transforma en *T_Asoc.op3.ind.*
Para las operaciones de *info_gpo* e *info_elem* los parámetros de éstas dos últimas primitivas son *tipo_primitiva* y *mensaje.*

En el caso de *destruye* son *tipo_primitiva* e *id_gpo.* El segundo se refiere al grupo que será destruido, y de ésta manera las E_SA remotas registren que sus respectivos usuarios ya no pertenecen a dicho grupo. Ésta última primitiva generada por la E_SA local es enviada a cada uno de los elementos del grupo.

Clase 4
La operación incluida dentro de éste clase tiene como finalidad mandar un mensaje de información a un agente. Ésta clase de operaciones será llamdo *op4* y está integrado de la siguiente manera.
op4:

info_elem	manda un mensaje informativo a un agente

El protocolo seguido por ésta es el siguiente y es ilustrado en la figura 5.9.

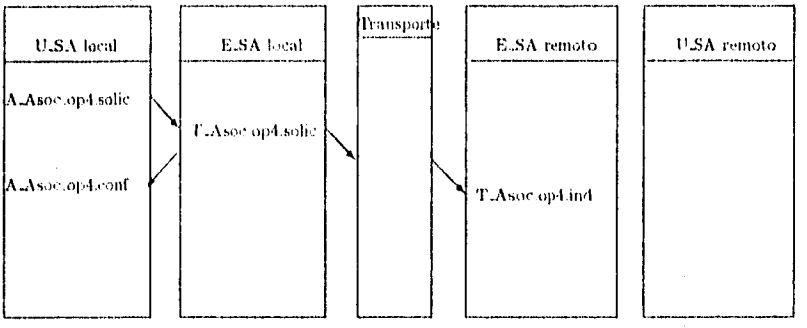


Fig. 5.9 Diagrama de secuencia temporal de primitivas de *op4*

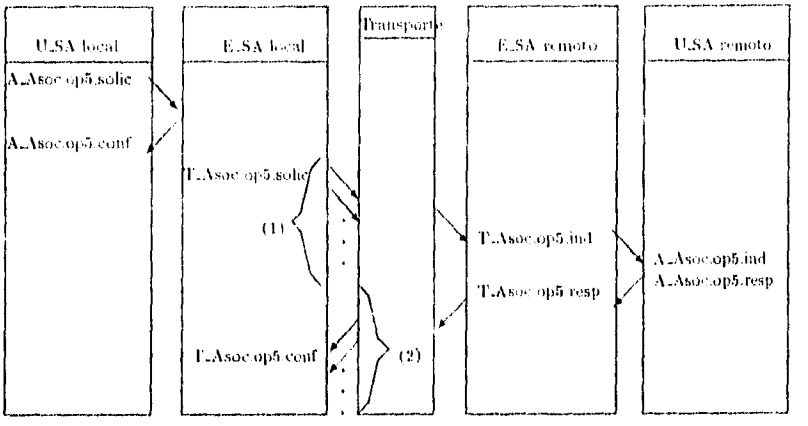
El U_SA local envía la primitiva *A_Asoc.op4.solic* a la E_SA local. Los parámetros de ésta son: *tipo_primitiva, id_elem, y mensaje.* El segundo es el identificador del agente al cual se requiere enviar el mensaje informativo, mismo que será puesto en el tercer parámetro.

Al recibir dicha primitiva la E_SA local genera *A_Asoc.op4.conf*, y es recibida por el U_SA local. El único parámetro de ésta primitiva es *resp.* En éste se le informa al usuario el código de error que ha generado la operación que solicitó.
Inmediatamente despues de enviar ésta primitiva la E_SA local genera *T_Asoc.op4.solic* y la envía a la E_SA remoto, la cual al recibirla se transforma en *T_Asoc.op4.ind.*
Los parámetros de éstas dos últimas primitivas son *tipo_primitiva* y *mensaje.*

Clase 5
La operación incluida en ésta clase tienen como finalidad hacer una solicitud a un grupo.
Ésta clase será llamada *op5* y está integrada de la siguiente manera.
op5:

solic_gpo	hace una solicitud a todos los elementos de un grupo

El protocolo seguido por éste grupo de operaciones es el siguiente y es ilustrado en la figura 5.10.



(1) n primitivas, donde n=numero integrantes del grupo

(2) m primitivas, donde
- m= numero de integrantes del grupo , sino se vence el timeout
- 0<=m<=numero de integrantes del grupo, si se vence el timeout

Fig. 5.10 Diagrama de secuencia temporal de primitivas *op5*

El U_SA local envía la primitiva *A_Asoc.op5.solic* a la E_SA local, cuyos parámetros son *tipo_primitiva, id_gpo, id_func, p_ent* y *p_sal*. El segundo es el identificador del grupo al cual se le solicita una función o servicio, mismo que es identificado a través del tercero.
p_ent y *p_sal* son los parámetros de entrada y de salida, respectivamente, de la función o servicio requerido.
Al recibir la E_SA local dicha primitiva genera a *A_Asoc.op5.conf* la cual es recibida por el U_SA local. El parámetro que acompaña a ésta primitiva es *resp*, el cual es el código de error de la operación realizada o en su defecto es el identificador de la solicitud, mediante el cual se podrá pedir posteriormente la o las respuestas de la función solicitada.

Inmediatamente después la E_SA local también envía la primitiva *T_Asoc.op5.solic* a la E_SA remota. Los parámetros que acompañan a ésta son: *tipo_primitiva, id_cliente, id_funcion, p_ent* y *p_sal*. Como se podrá observar los parámetros son similares a los que utiliza una de las primitivas mencionadas anteriormente con excepción del parámetro *id_cliente* que identifica al agente que solicita la ejecución de la operación en cuestión, éste parámetro servirá para que la E_SA remota determine si atenderá el servicio de quien se lo solicite y para que en el

caso afirmativo sepa a quién mandar la respuesta del servicio ofrecido. La E_SA local envía ésta primitiva a cada uno de los elementos que integren el grupo al cual se le hace la solicitud. Posteriormente arranca un timer que sirve para determinar el tiempo máximo que esperará para recibir contestaciones. También inicializa un contador que llevará la cuenta de las contestaciones recibidas, en el momento que el contador sea igual al número de elementos del grupo dejará de esperar más respuestas. Ésta primitiva al llegar a las E_SA remotas se convierte en *T_Asoc.op5.ind* y posteriormente en *A_Asoc.op5.ind* al ser recibida por los U_SA remotos.

A continuación la U_SA remota procede a enviar la primitiva *A_Asoc.op5.resp* cuyos parámetros son *resp* y *p_sal*. El primero es el código de error de la operación realizada y el segundo es el parámetro de salida que entrega dicha operación.
Al recibir la E_SA remota dicha primitiva, ésta genera a *T_Asoc.op5.resp* y que al ser recibida por la E_SA local se convierte en *T_Asoc.op5.conf*. Éstas dos últimas primitivas manejan los mismos pámetros que la mencionada antes de ellas.

La E_SA local recibe *T_Asoc.op5.conf* de todos los elementos del grupo que contestarón a su solicitud.
Y guarda el o los resultados que recibe de las E_SA remotas para que en el momento que el U_SA local pida una respuesta, ésta se le pueda entregar.

Clase 6

Las operaciones incluidas en ésta clase tienen como finalidad hacer una solicitud a un grupo o a un agente.
Éste grupo de operaciones será llamado *op6* y está integrado de la siguiente manera. *op6*:

solic_elem	hace una solicitud a un agente
solic_gpo_uno	hace una solicitud a un grupo para que sea atendida por uno de sus integrantes

El protocolo seguido por éste grupo de operaciones es el siguiente y es ilustrado en la figura 5.11.

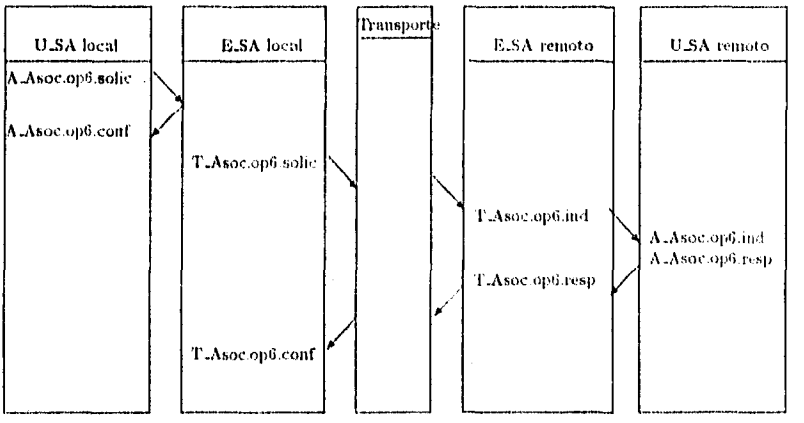


Fig. 5.11 Diagrama de secuencia temporal de primitivas *op6*

El U_SA local envía la primitiva *A_Asoc.op6.solic* a la E_SA local, si la operación es *solic_gpo_uno* los parámetros son *tipo_primitiva*, *id_gpo*, *id_func*, *p_ent* y *p_sal*. El segundo es el identificador del grupo al cual se le solicita una función o servicio, mismo que es identificado a través del tercero.
p_ent y *p_sal* son los parámetros de entrada y de salida, respectivamente, de la función o servicio requerido.
En el caso de la operación *solic_elem* los parámetros son los mismos a diferencia de *id_gpo*, ya que en lugar de éste se tiene *id_elem*. Éste identifica al elemento que se le está haciendo una solicitud. Al recibir la E_SA local dicha primitiva genera a *A_Asoc.op6.conf* la cual es recibida por el U_SA local. El parámetro que acompaña a ésta primitiva para las dos operaciones es *resp*, el cual es el código de error de la operación realizada o en su defecto es el identificador de la solicitud, mediante el cual se podrá pedir posteriormente la o las respuestas de la función solicitada.

Inmediatamente después la E_SA local también envía la primitiva *T_Asoc.op6.solic* a la E_SA remota. En el caso de la operación *solic_gpo_uno* la envía sólo a uno de los elementos del grupo que es elegido aleatoriamente.
Posteriormente arranca un timer que sirve para determinar el tiempo

máximo que esperará recibir la contestación. Los parámetros que acompñan a ésta primitiva son: *tipo_primitiva*, *id_cliente*, *id_funcion*, *p_ent* y *p_sal*. Como se podrá observar los parámetros son similares a los que utiliza una de las primitivas mencionadas anteriormente con excepción del parámetro *id_cliente* que identifica al agente que solicita la ejecución de la operación en cuestión, éste parámetro servirá para que la E_SA remota determine si atenderá el servicio de quien se lo solicite y para que en el caso afirmativo sepa a quién mandar la respuesta del servicio ofrecido.
Ésta primitiva al llegar a la E_SA remota se convierte en *T_Asoc.op6.ind* y posteriormente en *A_Asoc.op6.ind* al ser recibida por el U_SA remoto.

A continuación la U_SA remota procede a enviar la primitiva *A_Asoc.op6.resp* cuyos parámetros son *resp* y *p_sal*. El primero es el código de error de la operación realizada y el segundo es el parámetro de salida que entrega dicha operación.
Al recibir la E_SA remota dicha primitiva, ésta genera a *T_Asoc.op6.resp* y que al ser recibida por la E_SA local se convierte en *T_Asoc.op6.conf*. Éstas dos últimas primitivas manejan los mismos pámetros que la mencionada antes de ellas.

De ésta manera, para las dos operaciones de *op6*, la E_SA local guarda el resultado que recibe de la E_SA remota para que en el momento que el U_SA local pida una respuesta, ésta se le pueda entregar.

5.3.2 Especificación del protocolo del SA

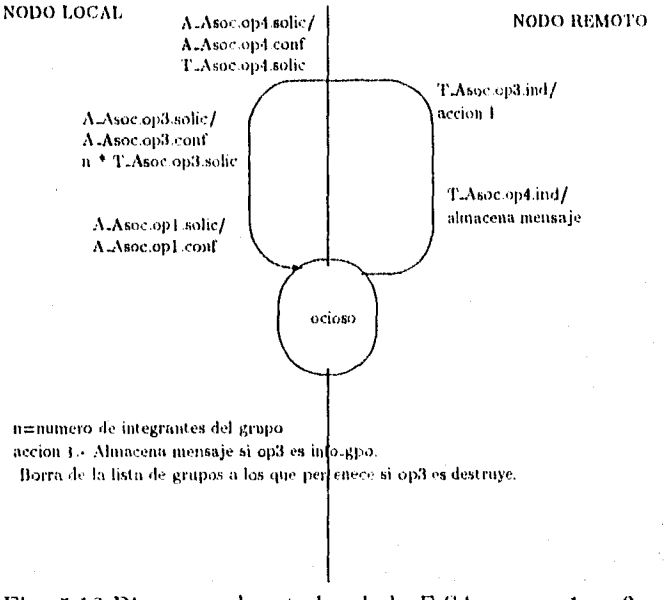


Fig. 5.12 Diagrama de estados de la E_SA para op1, op3, y op4.

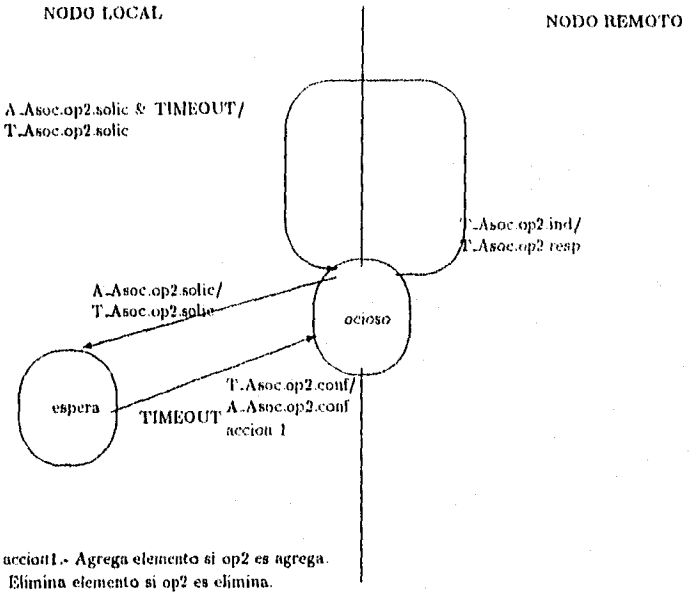
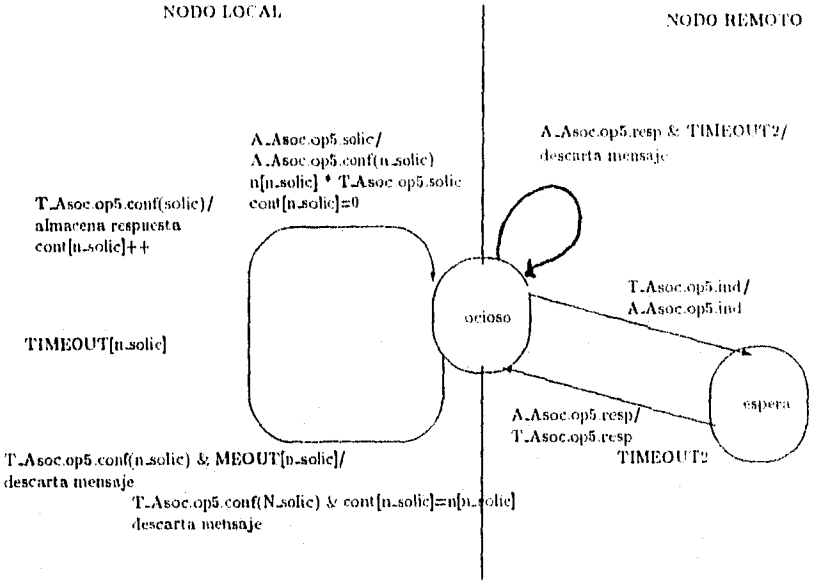
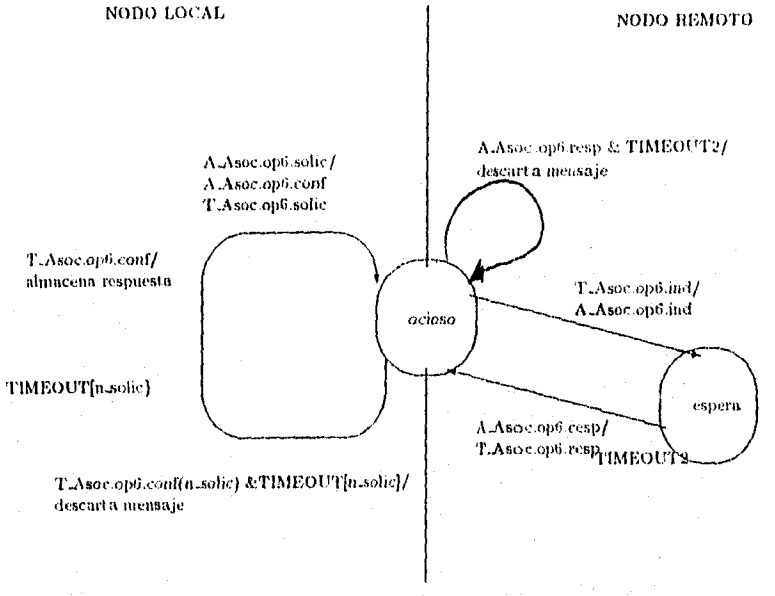


Fig. 5.13 Diagrama de estados de la E_SA para op2



n_solic.- numero asignado a una solicitud
n[]- arreglo que contiene el numero de integrantes de grupo correspondiente a una solicitud
cont[].- arreglo que contiene el numero de respuestas recibidas de una solicitud dada
TIMEOUT[].- arreglo de las vencimientos de las solicitudes realizadas

Fig. 5.14 Diagrama de estados de la E_SA para op5



n_solic.- numero asignado a una solicitud
TIMEOUT[].- arreglo de los vencimientos de las solicitudes realizadas

Fig. 5.15 Diagrama de estados de la E_SA para op6

Conclusiones

En este trabajo se ha presentado el diseño, la especificación y la implementación de la Interfaz de Comunicaciones (IC). Del Servidor de Acciones Privadas (SAP) sólo se presentó su diseño y una pequeña implementación. El primero se encarga de proporcionar los servicios básicos que permiten que los agentes se comuniquen entre ellos.
El segundo consta de servicios más elaborados y que son construidos a partir de los que son provistos por la IC.

En cuanto a los objetivos planteados en la introducción:

1. La utilización de la arquitectura del *facilitador*.

2. La integración de los servicios distribuidos existentes.

3. El manejo de grupos dinámicos de trabajo, los cuales consideramos son de gran ayuda para el trabajo cooperativo.

4. La utilización de un servicio que permita la localización de un usuario en la red de comunicaciones de un sistema distribuido.

se obtuvieron los siguientes resultados:

Se utilizó como base para el desarrollo del presente trabajo la arquitectura del *facilitador*, misma que nos proporcionó una base para el desarrollo de la tesis.
Sin duda alguna podemos decir que ésta plataforma fue de gran utilidad, ya que nos permitió concentrar nuestro trabajo en objetivos bien definidos, es decir, en el desarrollo del SAP y de la IC.

Por otro lado, también se aprovechó la integración de servicios distribuidos existentes proporcionados por una red TCP/IP-Unix, y son los de: transferencia remota de archivos (ftp), correo electrónico (mail), y la ejecución remota de procesos (rsh).
Lo anterior facilitó mucho el desarrollo del trabajo, y permitió, además que éste fuera más completo y robusto.

En un principio sólo se pretendía el uso de grupos dinámicos sin ninguna jerarquía. Sin embargo, en la evolución del trabajo nos dimos cuenta que era mejor si incluíamos el concepto de patrón evolutivo de las organizaciones, expuesto por Fox [S.80]. De ésta manera, se decidió diseñar una estructura que en lugar de que sólo representara grupos simples, fuera capaz de representar grupos con diferentes grados organizacionales.
Creemos, al igual que Gasser [LCN88], que los sistemas cooperativos requieren de una plataforma que soporte la capacidad de representar organizaciones flexibles.
Generalmente la problemática de la organización de un sistema cooperativo no es abordado de manera directa, sin embargo, nosotros consideramos que éste aspecto debe ser uno de los puntos de la estrategia a seguir en el diseño de tales sistemas, ya que un trabajo cooperativo no puede prescindir de una organización.
Aunque desafortunadamente, por cuestiones de tiempo, sólo se implementaron las operaciones que permiten el uso de grupos sin jeraquía.
Sin embargo, sí se definió la propuesta para grupos multiorganizacionales y ésta puede ser aprovechada por algún trabajo futuro, como base para el desarrollo de éste tipo de grupos.

También se pretendía en un principio el desarrollo de un servicio que proporcionara la ubicación de un usuario. A través del desarrollo del trabajo nos dimos cuenta que ésta era una visión muy cerrada, y que lo que se debería pretender es la elaboración de un servicio que proporcionara la ubicación de cualquier entidad atada a la red de comunicaciones de un sistema cooperativo.
De ésta manera, el diseño de dicho servicio ofrece la capacidad de localizar usuarios, grupos, y recursos.
El servicio de localización es indispensable para la formación dinámica de grupos y para la identificación de cualquier elemento que se encuentre conectado a la red. Dicho elemento puede ser un dispositivo, como una impresora, un lector de cintas, etc. También puede ser un agente, o incluso algún servicio que éste ofrezca.

Además se introdujo, fuera de los objetivos iniciales, el manejo de patrones en el proceso de localización de un ente en la red. Se decidió

hacer esto debido a que el uso de patrones permite que la ubicación e identificación sea llevada a cabo mediante características parciales y no solamente a través del identificador único de cada ente.
Tal herramienta de localización es muy flexible y poderosa para la búsqueda de cualquier elemento conectado a la red de comunicaciones. Con el servicio de localización (SL) se puede ubicar de manera eficiente al elemento, que en un dado momento se necesite, y que mejor cumpla con las características requeridas.
Y no solamante es útil en búsquedas, sino también el la recepción y transmisión de información, y en la asignación de tareas.
Puede mandarse información a las entidades que cumplan con ciertas características, como por ejemplo mandar un correo electrónico, que contiene información sobre un simposium de sistemas expertos, a todos los usuarios que ésten interesados en la Inteligencia Artificial.
Otra de las ventajas del uso de ésta herramienta es asignar una tarea al usuario que cumpla con las características necesarias para que la pueda realizar. En éste caso no importa quien la ejecute, sino tan sólo que ésta sea realizada.

También por cuestiones de tiempo, el SL no fue implementado. Sin embargo, se definió la propuesta e incluso su especificación, mismas que pueden se aprovechadas como un antecedente en el desarrollo de trabajos futuros.

En cuanto a lo que se realizó de la implementación, la operación que más trabajo costó desarrollar fue la de *solicitar*.
Lo anterior fue debido a que la propuesta para dicha operación era de que no fuera bloqueante, es decir, que el usuario de dicha operación, al hacer uso de ella no se quedara esperando la respuesta, sino que continuara realizando otras actividades mientras se resuelve su solicitud.

Ésta decisión se tomó por la razón de que hay solicitudes que pueden tardarse mucho tiempo en realizarse, como puede ser la multiplicación de matrices grandes.

Tal operación fue implementada con RPCs ya que éstas ofrecen la gran ventaja de hacer el paso de parámetros de manera transparente. Si en lugar de éstas hubieramos usado *sockets*, tendría que haberse con-

struído un módulo para la transportación de los parámetros de entrada y de salida a través de la red, lo cual no es trivial.

Sin embargo, el problema al cual nos enfrentamos es que las RPCs son bloqueantes, cosa que deseabamos evitar.
Para solucionar tal problema se crea un proceso *p_espera* que se encarga de realizar la RPC y por lo tanto se queda bloqueado hasta que le envían la respuesta.

El usuario del *servicio de asociación* (U_SA), pide la respuesta con la operación *pide_rta*, la cual, a su vez, hace un RPC al *p_espera*, sin embargo, éste no se queda bloqueado ya que si no contesta en un periodo de un segundo la RPC regresa, caso en el cual el U_SA infiere que el *p_asoc* aún no tiene la respuesta.

La implementación se ha realizado en C++, sin embargo, se propone como trabajo futuro la reprogramación de los módulos en un lenguaje concurrente orientado a objetos.
Esto es debido a que, como se mostró en la sección 2.9, este tipo de lenguajes son ideales para la programación de sistemas de (IAD), por poseer características tales como encapsulación, control local, comunicación basada en mensajes, y objetos de multigranularidad heterogénea.

El trabajo realizado en ésta tesis presenta una propuesta de un sistema distribuido como plataforma para soportar la formación, comunicación e interacción entre grupos cooperativos. Ésta colabora en nutrir todas aquellas que ya han sido desarrolladas y que están por desarrollarse.

Los sistemas cooperativos son una tecnología emergente que apenas está viendo crecer sus primeros frutos, falta aún mucho por desarrollar. Éstos sistemas forman parte del proceso de globalización de las comunicaciones que en la actualidad padece el mundo.
A través de ésta se podrá tener acceso, desde una interfaz común, a todo tipo de recursos y de información, y además será posible la cooperación entre cualquier clase de agentes.

Apéndice I

Implantación

I. Implantación

La implantación de la IC y del SAP se llevó a cabo en una Sun Sparc Station 10, con sistema operativo Unix BSD, versión 4.1.3. Ésta se encuentra conectada a una red Unix-TCP/IP, y la programación se realizó en C++: utilizando el compilador de GNU versión 2.5.8. La interfaz gráfica de la demo fue construida con *Open Look*.

Los archivos principales que fueron desarrollados son los siguientes:

- Para la Demo el archivo principal es "p.c"
- El manejo de grupos se desarrolló con los siguientes archivos:

  1. main_grupo.h
  2. grupo.h
  3. grupo.c

  El primero es el programa principal. En el segundo se encuentra la definición de la clase "Grupo" y de sus métodos. En el tercero se encuentra la implementación de éstos.

- El módulo de la IC se encuentra definida en los archivos:

  1. main_asoc.c
  2. asociacion.h
  3. asociacion.c

  El primero es el programa principal. El segundo define a la clase y los métodos de la IC. Y el último tiene la implementación de los métodos.

- Las librerías de las operaciones implementadas se encuentran en:

  1. lib_ic.h
  2. lib_sap.h

  En el primero se encuentran las operaciones definidas para el acceso a la IC. En el segundo se encuentran aquellas que accesan al SAP.

II. Implantación del SAP

En cuanto al SAP, se han implementado dos tareas, las cuales utilizan los servicios definidos por la IC. La estructura de éstas se ilustra a continuación.

Tarea 1

Descripción
Agrega los agentes "xico" y "cardel" al grupo "papantla1". El agente que invoque ésta operación debe encontrarse en "papantla.lania.mx", y dicho agente debió de haber creado previamente el grupo mencionado.

Sintaxis: int tarea1()

Regresa:

- *0.-* Éxito.
- *-1.-* Error, no pudo agregar uno o ninguno de los agentes.

Tarea 2

Descripción
Crea el directorio "/tmp3/" en el host "xico.lania.mx", y transfiere el archivo ".cshrc" del host donde se encuentra el agente que invoca la operación al directorio recien creado.

Sintaxis:
*char *tarea2()*

Regresa:
Los códigos que los servicios de *ejecución remota* y de *transferencia de archivos* devuelven, son los que se entregan como resultado.

III. Implantación de la IC

La IC se encuentra básicamente compuesta por:

1. el servicio de correo
2. el servicio de transferencia de archivos
3. el servicio de ejecución de tareas remotas
4. el servicio de Asociación
5. el servicio de Localización

De éstos servicios, se encuentran implementados sólo los cuatro primeros. Debido a que del conjunto de servicios que ofrece una red TCP/IP-Unix ya se encuentran implementados los tres primeros, y por lo tanto se decidió no volver a implementarlos.
Por ésta razón, sólo se desarrollaron las operaciones que nos permiten hacer el llamado a estos servicios. Dichas operaciones tienen la siguiente estructura:

Transferencia de archivos

Descripción:
Permite la transferencia de archivos remotos. Se pueden transportar de un host remoto a otro local y viceverza.

Sintaxis:
*char *ftp(char *host,char *usuario,char *passwd,char *modo, char *operac,char *arch_loc,char *arch_rem)*

Parámetros:

- host.- Dirección IP o nombre completo del host remoto.
- usuario.- Es el *login* que el usuario posee en el host remoto.
- passwd.- Es la contraseña que el usuario posee en el host remoto.
- modo.- Los modos pueden ser:
    1. ascii.- Permite transferir archivos de texto.

2. binary.- Permite transferir archivos binarios.

- operac.- Las operaciones pueden ser:

  1. get.- Transfiere un archivo del host remoto al local.
  2. put.- Transfiere un archivo del host local al remoto.

- arch_loc.- Es el archivo local con su *path* completo.
- arch_rem.- Es el archivo remoto con su *path* completo.

Regresa:
El código de error que regresa es el mismo que proporciona el servicio de *ftp* definido en una red TCP/IP-Unix.

Correo

Descripción:
Hace uso del correo electrónico, el cual permite mandar correo a usuarios remotos.

Sintaxis:
*char *correo(char *host,char *usuario,char *arch)*

Parámetros:

- host.- Dirección IP o nombre completo del host donde se encuentra el ususario al cual se requiere enviarle un correo.
- usuario.- Es el id o *login* del usuario remoto.
- arch.- Es el archivo que contiene la información que se desea enviar.

Regresa:
El código de error que regresa es el mismo que proporciona el servicio de *mail* definido en una red TCP/IP-Unix.

Ejecución remota

Descripción:
Permite la ejecución remota de programas.

Sintaxis:
*char *rsh(char *host,char *comando)*

Parámetros:

- host.- Es el nombre del host o dirección IP de la máquina donde se desea ejecutar el comando.
- comando.- Es el comando a ejecutar remotamente.

Regresa:
El código de error que regresa es el mismo que proporciona el servicio de *rsh* definido en una red TCP/IP-Unix.

Por otro lado, el *servicio de asociación* sí fue implementado por completo. A continuación se describe la implementación de éste servicio.

IV. Implantación del servicio de Asociación

Como ya se mencionó anteriormente, las operaciones del *servicio de asociación* que fueron implementadas son:

- Crear un grupo (crea_gpo)
- Destruir un Grupo (destruye)
- Agregar un elemento (agrega)
- Eliminar un elemento (eliminar)
- Consultar:
  - Si está un elemento determinado (esta_elem)
  - El número de elementos (num_elem)

  - Grupos de los que soy dueño (gpos_dueño)
  - Grupos en los que participo (gpos_participo)

- Habilitar el uso remoto de una función (habilita)
- Deshabilitar el uso remoto de una función (deshabilita)
- Informar a un grupo (informa_gpo)
- Informar a un agente (informa_agente)
- Obtener el primer mensaje de información recibido sin leer (dame_info)
- Solicitar a un grupo múltiples respuestas (solicita_gpo)
- Solicitar a un grupo una respuesta (solicita_gpo_uno)
- Solicitar a un agente (solicita_agente)
- Pedir una respuesta (dame_respuesta)
- Terminar una solicitud (termina_solicitud)
- Registrar un agente (pon_agente)
- Eliminar del registro un agente (quita_agente)
- Registrar una función (pon_función)
- Eliminar una función (quita_función)

Las últimas 4 operaciones no están definidas en el diseño de éste servicio, sin embargo, han sido agregadas por considerarse necesarias para la correcta operación del *servicio de asociación*. Más adelante se da la explicación de cada una de ellas.

El *usuario del servicio de asociación* (U_SA) se comunica con la *entidad del servicio de asociación* (E_SA) a través de un socket del dominio unix con datagramas.

La E_SA está integrada por el siguiente conjunto de procesos:

- proceso de asociación (p_asoc)
- proceso de grupo (p_gpo)
- proceso de espera (p_espera)

El *p_asoc* es el principal, éste atiende las llamadas que le hace el U_SA. Tiene definido cuatro sockets:

1. ux
2. inet_rx
3. inet_ack
4. inet_tx

El primero es del dominio Unix con datagramas y es usado para recibir las llamadas del U_SA. El segundo es del dominio Internet con datagramas y opera en el puerto 7000 para recibir mensajes provenientes de otros agentes a través de la red.
El tercero es igual que el anterior, sólo que está definido en el puerto 5000. Éste se utiliza para recibir acuce de recibo o contestación de alguno o algunos agentes remotos.
El último es del dominio Internet con datagramas y se utiliza para mandar mensajes a la dirección y puerto que sean necesarios.

Cuando se requiere que un grupo sea creado el *p_asoc* crea un proceso *p_gpo* utilizando las llamadas al sistema *fork* y *exec*. Éste tiene definido tres sockets pertenecientes al dominio Internet con datagramas:

1. inet_rx
2. inet_ack
3. inet_tx

El primero tiene asignado un puerto a partir del 7001 que no esté ocupado, y es utilizado para recibir mensajes provenientes de la red.
El segundo tiene asignado un puerto a partir del 5000 que no esté

ocupado, y es utilizado para recibir acuse de recibo o contestación de alguno o algunos agentes.
Al tercero se le asigna una dirección y un puerto de manera dinámica, y es utilizado para mandar mensajes a través de la red.

En el caso de que se requiera hacer una solicitud a un grupo o a un agente se crea un proceso *p_espera* para que sea éste quien se quede esperando la respuesta de dicha solicitud y de ésta manera no quede bloqueado ni el U_SA ni el proceso *p_asoc*.
El *p_espera* utiliza RPCs de bajo nivel para realizar la solicitud y para entregar el resultado al U_SA.

Los procesos interactúan de la siguiente manera:

- U_SA local - p_asoc local.- a través de un socket del dominio unix con datagramas
- U_SA local - p_espera local.- a través de RPCs
- p_asoc local - p_asoc remoto.- a través de un socket del domino Internet con datagramas
- p_asoc local - p_grupo.- a través de un socket del dominio Internet con datagramas
- p_espera local - U_SA rmoto.- a través de RPCs

V. Intercambio de mensajes

A continuación se describe el paso de mensajes que se lleva a cabo entre los procesos mencionados anteriormente para cada una de las operaciones implementadas.
Aquello que esté entre comillas en las Unidades de Datos del Protocolo (PDU, con sus siglas en inglés) de los mensajes significa una cadena de caracteres.

crea_gpo

En la figura 1.1 se ilustra el esquema seguido por la operación *crea_gpo*.

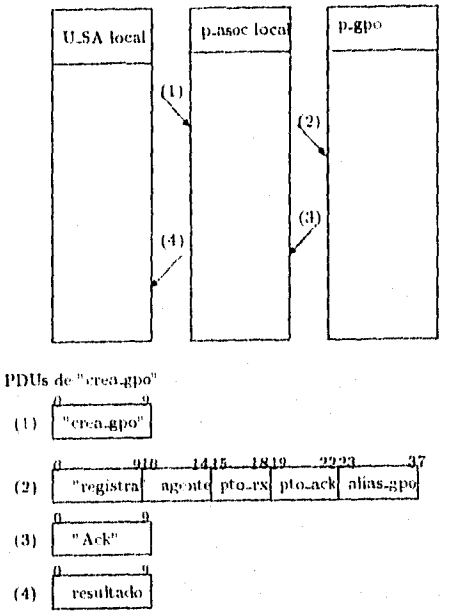


Fig. I.1 Intercambio de mensajes para la operación crea_gpo

Al recibir el *p_asoc* el mensaje (1), crea el proceso *p_gpo* y le manda a (2) con el fin de que registre a su dueño indicandole en éste su identificador, el puerto por donde recibe mensajes y por donde recibe el *acknowledge*. Además le indica cuál es su identificador de grupo.
A ésto el *p_gpo* contesta con un *acknowledge*, enviando a (3) al puerto *pto_ack* del agente dueño.
El *p_asoc* espera recibir tal contestación en un periodo no mayor a 5 segundos. Al recibirlo o vencerse el *timeout* contesta al U_SA el resultado de la operación con (4).

Cada grupo tiene registrado a cada uno de sus integrantes en una lista ligada (lista_elem) con la información de id, dirección IP, puerto y rol. El id es el identificador del elemento, el cual está constituido por una cadena de caracteres. La dirección IP es aquella donde se localiza, y el puerto es aquel por donde recibe mensajes, es decir es el puerto *inet_rx* del integrante.

Análogamente cada agente tiene una lista de:

- los grupos que es dueño (lista_gpos_dueño)
- los grupos de los cuales es integrante (lista_gpos_participo)
- mensajes de informacion recibidos (lista_info)

Cada elemento de las dos primeras listas está integrada por:
id del grupo, dirección IP del grupo, y puerto *inet_rx* del grupo.
En la tercera cada elemento es un mensaje informativo que se ha recibido.

Adicionalmente se tienen dos archivos para el registro de las funciones y de los agentes:

- "funciones.dat"
- "agentes.dat"

En el primero se tiene la relación de las funciones que el usuario local y los remotos ofrecen. Y está integrado de la siguiente manera:
id de la función, y número de función.

El segundo archivo tiene la información referente a los agentes, y su formanto es:
id del agente, dirección IP donde reside, y su puerto *inet_rx*.
Cada agente tiene, por separado, las listas y archivos que han sido mencionados anteriormente.

agrega

En la figura I.2 se ilustra el esquema seguido por la operación *agrega*.

El *p_asoc local* recibe el mensaje (1) y de aquí obtiene el identificador del grupo y del cliente.
A través de dicho id puede obtener la dir IP y el puerto *inet_rx* del agente que se desea agregar.
Posteriormente, el *p_asoc_local* arranca un *timer* de espera y envía a (2) indicando el id, host y puerto del agente solicitante y del elemento que se desea agregar. También envía el puerto por donde esperará la confirmación por un tiempo determinado, mismo que al vencerse regresaría un código de error en (6).
El *p_gpo* arranca también un *timer* de espera y le indica con (3) al *p_asoc remoto* el grupo al cual será agregado y así éste último pueda registrarlo en su lista *lista_gpos_participo*.

Al recibir *p_gpo* a (4) registra en *lista_elem* la inclución del elemento, y envía a (5), mismo que al recibirlo el *p_asoc local* envía finalmente a (6) a el USA.

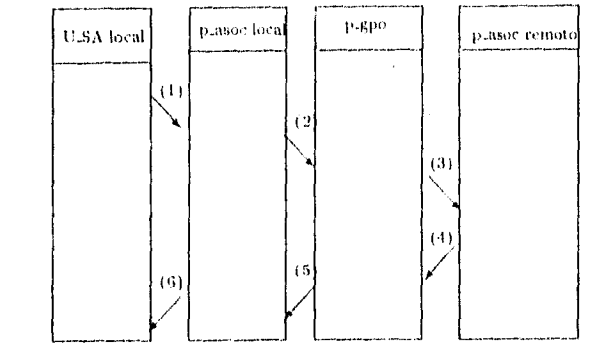
elimina

En la figura I.2 se ilustra el esquema seguido por la operación *elimina*.
Éste es muy similar al de la operación anterior. Las únicas diferencias son que al recibir a (3) el *p_asoc remoto*, borra al grupo de la lista *lista_gpos_participo*.
Y el *p_gpo* al recibir a (4) borra el elemento de su lista *lista_elem*.

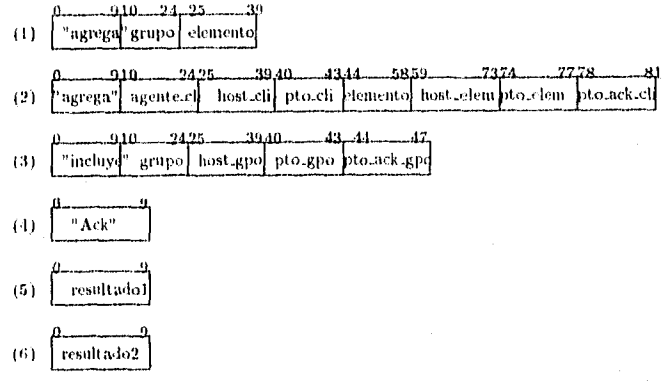
esta_elem

En la figura I.3 se ilustra el esquema seguido por la operación *esta_elem*.

El *p_asoc local* recibe a (1) y de éste obtiene el identificador del grupo y del elemento que se desea verificar si se encuentra. A continuación el *p_asoc* arranca un *timer* de espera y envía a (2), del cual el *p_gpo* obtiene el id, la dirección IP y el puerto del agente cliente y del elemento.



PDUs de "agrega":



PDUs de "elimina":

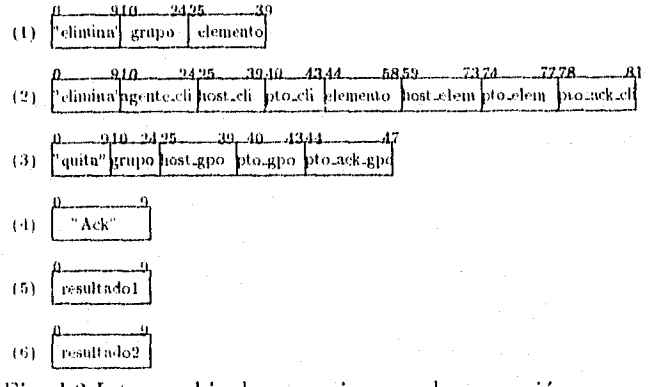
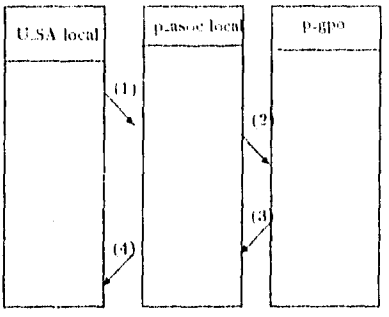
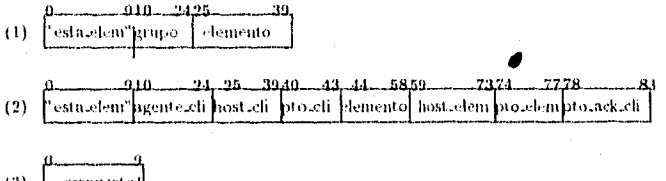


Fig. I.2 Intercambio de mensajes para la operación agrega y elimina



PDUs de "esta_elem":



PDUs de "num_elem":

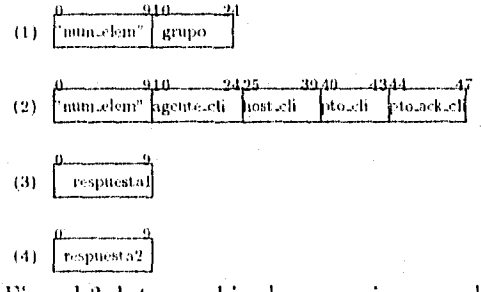


Fig. 1.3 Interrambio de mensajes para las operaciones esta_elem y num_elem

También recibe el puerto por donde deberá confirmar la respuesta con el mensaje (3) y por último el *p_asoc local* entregará a el U_SA el resultado de la respuesta de la operación enviandole a (4).

num_elem

En la figura I.2 se ilustra el esquema seguido por la operación *num_elem*.
El *p_asoc* recibe a (1) y de éste obtiene el id del grupo al cual se le preguntará el número de elementos que contiene.
Para ésto arranca un *timer* de espera y genera a (2) de donde el *p_gpo* obtiene el id, la dir IP y puerto del agente cliente, además del puerto por donde espera la respuesta éste último.
Por consiguiente el *p_gpo* contesta con (3) y a su vez el *p_asoc local* con (4).

destruye

En la figura I.4 se ilustra el esquema seguido por la operación *destruye*.

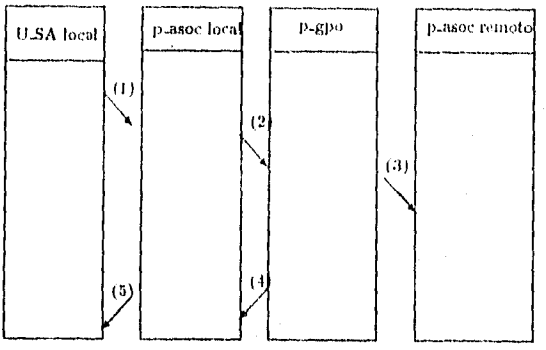
El *p_asoc local* recibe a (1), el cual le indica el id del grupo que se desea destruir. Para esto envía a (2) al *p_gpo* correspondiente y arranca un *timer* que determina el tiempo que se esperará la confirmación.
Al recibir *p_gpo* a (2) obtiene el id, la dir IP, el puerto *inet_rr* y el *inet_ack* del cliente, y verifica si éste es dueño del grupo. En caso de que lo sea, envía a (3) a cada uno de sus integrantes, los cuales al recibirlo eliminan de su lista *lista_gpos_participo* al grupo correspondiente.

El *p_gpo* también genera (4) para confirmarle al *p_asoc local* el resultado de la operación, y posteriormente se autodestruirá.
En consecuencia el *p_asoc local* envía a (5) al U_SA local, indicandole el resultado de la operación.

gpos_participo y gpos_dueño

En la figura I.5 se ilustra el esquema seguido por las operaciones *gpos_participo* y *gpos_dueño*.

Al recibir el *p_asoc local* a (1) revisa su lista *lista_gpos_participo* o *lista_gpos_dueño*, si la operación es *gpos_participo* o *gpos_dueño*, respectivamente.



PDUs de "destruye".

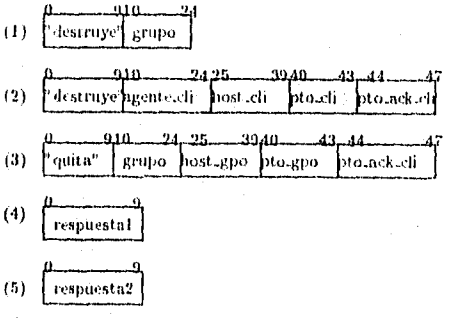


Fig. I.4 Intercambio de mensajes para la operación destruye

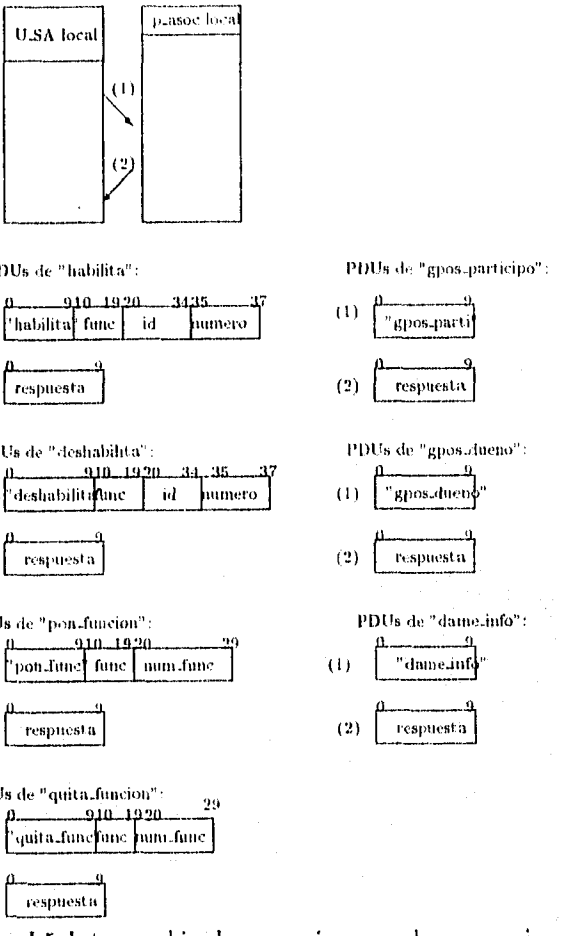


Fig. I.5 Intercambio de mensajes para las operaciones *gpos_participa*, *gpos_dueño*, *habilita*, *deshabilita*, *dame_info*, *quita_func*, y *pon_func*.

Posteriormente contesta con la información requerida en (2).

habilita y deshabilita

En la figura I.5 se ilustra el esquema seguido por las operaciones *habilita* y *deshabilita*.
En los dos casos el *p_asoc local* recibe a (1) y de éste obtiene la función que se desea deshabilitar o habilitar, según sea el caso, para un grupo o agente del cual se hace referencia en el campo de "id".
Se podrá aumentar o disminuir el número de veces a que tendrá derecho a acceder a la función , lo cual es indicado en el último campo.

Tal registro de permisos se lleva en una lista (lista_permisos) de los procedimientos en donde cada elemento de ésta es una lista de los identificadores de los elementos o grupos que tiene derecho a accesarla. La lista 1 pertenece al procedimiento 1, la lista 2 al procedimiento 2, y asi sucesivamente. Por ejemplo, si un elemento tiene 3 permisos para accesar la función, se encontrará tres veces el id de éste en la lista correspondiente.

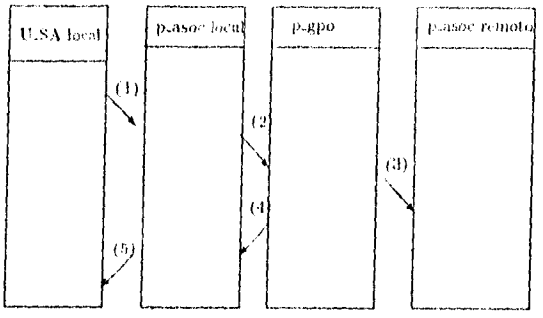
dame_info

En la figura I.5 se ilustra el esquema seguido por la operación *dame_info*.
El *p_asoc local* recibe a (1) y en consecuencia revisa la lista *lista_info*, desprendiendo de ésta el primer mensaje. Éste es enviado al *U_SA local* a través de (2).

quita_func y pon_func

En la figura I.5 se ilustra el esquema seguido por las operaciones *quita_func* y *pon_func*.
Al recibir el *p_asoc local* a (1) obtiene, del campo de func, el nombre de la función que desea agregar o eliminar, según sea el caso.
A través del último campo obtiene el número de función.
Ésta información es agregada o eliminada, según sea el caso, de un archivo ("funciones.dat") en el cual se tienen registradas las funciones con su respectivo número de programa.
Posteriormente el *p_asoc local* contesta al *U_SA local* con el resultado de la operación mediante (2).

informa_gpo



PDUs de "info_gpo":

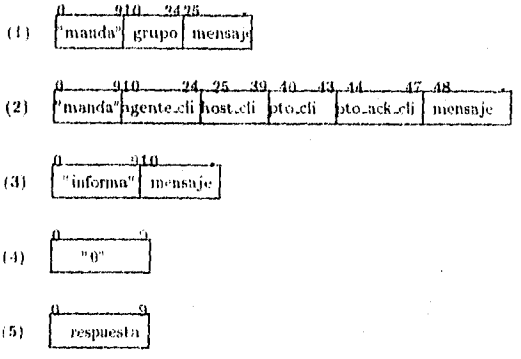


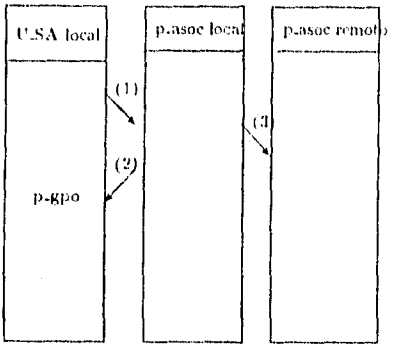
Fig. I.6 Intercambio de mensajes para la operación informa_gpo

En la figura I.6 se ilustra el esquema seguido por la operación *informa_gpo*.

Al recibir el *p_asoc local* a (1) obtiene al grupo y el mensaje que se desea enviar. Para ésto arranca un *timer* para la espera de la confirmación y genera a (2) de donde el *p_gpo* obtiene el id , dirección IP, puerto *incl_rx*, el puerto *incl_ack* del agente cliente y finalmente el mensaje.
Posteriormente envía a (3) a cada uno de sus integrantes y también a (4) para confirmar al *p_asoc local* el resultado de la operación. Éste último a su vez hace lo mismo con el U_SA local enviendole a (5).

informa_uno

En la figura I.7 se ilustra el esquema seguido por la operación *informa_uno*.



PDUs de "info_uno":

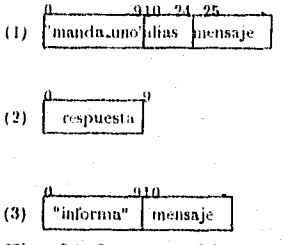


Fig. I.7 Intercambio de mensajes para las operación informa_uno

Al recibir *p_asoc local* a (1) obtiene el identificador del agente y el mensaje que se le desea enviar.
Como consecuencia envía a (2) al U_SA indicandole el resultado de la operación. Posteriormente manda a (3), mismo que al ser recibido por el *p_asoc remoto* lo almacena en su lista *lista_info*.

Solicitar

Cada agente puede ofrecer realizar ciertas funciones (por ejemplo, multiplicar matrices).
En éste caso, se implementarón las funciones de suma, resta, multiplicación y división de dos enteros, las cuales se llevaron a cabo con RPCs de bajo nivel, ya que sólo mediante éstas puede definirse un determinado tiempo de espera en cada llamada a un procedimiento remoto.
Las operaciones implementadas que implican una solicitud son las siguientes:

- solicita_agente
- solicita_gpo
- solicita_gpo_uno

Debido a que las operaciones que se encargan de realizar solicitudes no son bloquentes y de que las RPCs si lo son, se buscó la manera de que no se bloquera el U_SA local, ni el *p_asoc local*.
Para lograr ésto, cada vez que el U_SA invoca alguna de las operaciones para realizar una solicitud se crea un *p_espera* para cada solicitud realizada. El *p_espera* realiza el llamado del procedimiento remoto que se encuentra definido por el U_SA remoto. De manera que el *p_espera* es quien se queda bloqueado esperando la respuesta.

El U_SA local puede pedir tal respuesta con la operación *pedir_rta*. Para ésto hace un RPC al *p_espera* solicitandole la respuesta. Sin embargo, si el *p_espera* aún sigue bloquedo esperando la respuesta, la RPC invocada por el U_SA remoto regresa inmediatamente indicando que *p_espera* no contestó, y en éste caso se infiere que aún no tiene la respuesta. Ésto se logró fijando el tiempo de espera muy pequeño para dicha RPC.

Cada RPC tiene definido:

- un número de programa
- un número de versión
- un número de procedimiento

Para uniformizar y facilitar el manejo de las RPCs entre los agentes se decidió fijar un sólo número para el programa y otro para la versión en todos los agentes, y son los siguientes:

- número de programa: 0x20000989
- número de versión: 1

El número de procedimiento también sigue cierta uniformidad, ya que cada tipo de función tendrá un sólo número de procedimiento para todos los agentes.
De ésta manera todos los agentes tienen definido el número de procedimiento de la manera siguiente:

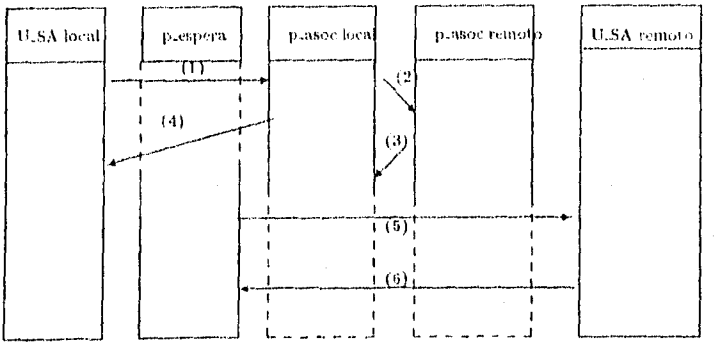
- suma: 1
- resta: 2
- multiplicación: 3
- división: 4

La relación anterior se encuentra definida en el archivo "funciones.dat" para cada agente. Los cuales tendrán que actualizar dicho archivo al agregar o eliminar una función, utilizando las operaciones *pon_func* o *quita_func*, según sea el caso.

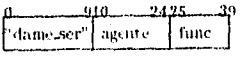
solicita_agente

En la figura I.8 se ilustra el esquema seguido por la operación *solicita_agente*.

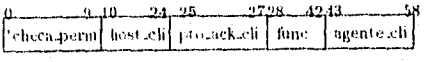
El U_SA envía a (1) al *p_espera*, donde éste último obtiene el id del agente remoto y la función que se le solicitará.
En consecuencia *p_espera* arranca un *timer* para esperar la confirmación



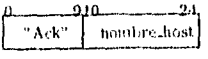
PDUs de "solicita_elem":



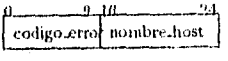
(1)



(2)



(3)



(4)

(5) Llamada a RPC: nombre host, direccion del procedimiento,timeout, p_ent, p_sal

(6) Contestacion de RPC: codigo de error, p_sal

Fig. 1.8 Intercambio de mensajes para las operación solicita_agente

del mensaje (2) que envía al *p_asoc remoto*, mismo que al recibirlo obtiene la dirección , el id y el puerto *inet_ack* del agente cliente, además del id de la función solicitada.

De ésta manera el *p_asoc remoto* investiga el número de la función en el archivo "funciones.dat" y verifica en la lista *listas_permisos* si tiene permiso de accesarla. En caso afirmativo, disminuye en uno sus permisos (elimina una vez el id del agente de dicha lista) y confirma la respuesta con (3). En el caso negativo, no envía a (3).

De ésta manera el *p_asoc local* obtiene el nombre del host del U_SA remoto y a su vez el *p_asoc local* contesta con (4) a el U_SA, obteniendo éste último, el código de error de la operación y en caso de ser exitosa el nombre del host al cual realizará la RPC.

Para ésto, crea con la llamada al sistema *fork* al *p_espera*, quien se encargará de realizar dicha RPC con (5), indicando el host, la dirección del procedimiento (número de programa, de versión y de procedimiento), el tiempo máximo que se esperará la respuesta, los parámetros de entrada y de salida. Y con (6) obtiene un código de error y los parámetro de salida.

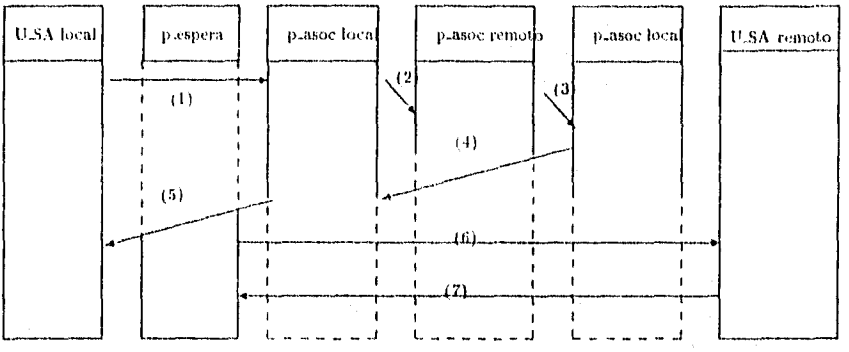
solicita_gpo

En la figura I.9 se ilustra el esquema seguido por la operación *solicita_gpo*.

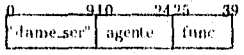
El procedimiento es casi idéntico al seguido por la operación anterior, la diferencia es que en lugar se hacerse la solicitud a un elemento se la hace al grupo, el cual se encarga de preguntar a todos sus elementos si se tiene permiso para solicitar la función de que se trate. Para ésto envía a (3) a cada uno de ellos, los cuales al recibirlo realizan lo mismo que en la operación anterior, contestando sólo aquellas que verificarón que si tiene permiso.

Para cada una de éstas contestaciones el *p_asoc local* recibe un mensaje del tipo (4).

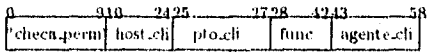
Posteriormente forma una lista de aquellos que tuvo contestación y la envía al U_SA local con (5). Éste último crea, asignandoles un número de programa diferente, un *p_espera* por cada elemento de la



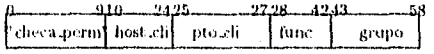
PDUs de "solicita_elem":



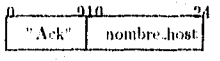
(1)



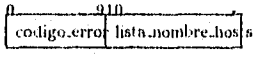
(2)



(3)



(4)

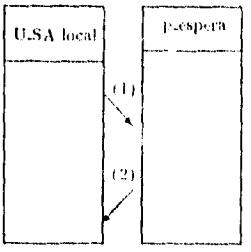


(5)

(6) Llamada a RPC: nombre host, direccion del procedimiento,timeout, p_ent, p_sal

(7) Contestacion de RPC: codigo de error, p_sal

Fig. I.9 Intercambio de mensajes para las operación solicita_gpo y solicita_gpo_uno



PDUs de "pide_rta"

(1) Llamada al RPC: nombre del host, direccion del procedimiento, timeout, parametros de salida

(2) Contestacion del RPC: codigo de error, parametros de salida

Fig. I.10 Intercambio de mensajes para las operación pide_rta

lista que se encargan de realizar las RPCs a los U_SA remotos que les correspondan enviando a (6) y obtendrán cada uno de ellos la respuesta recibiendo un mensaje (7).
De cada *p_espera* que se crea se guarda en una lista su número de programa y ésta se inserta en una lista de solicitudes (listas_solic).
Es de aquí donde se obtendrán las referencias (número de programa) de las RPCs para pedir el resultado de una solicitud determinada.

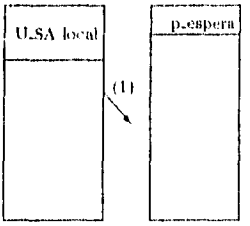
solicita_gpo_uno

En la figura I.9 se ilustra el esquema seguido por la operación *solicita_gpo_uno*.
El procedimiento de ésta operación es muy similar al seguido por *solicita_gpo*. La diferencia es que en éste caso al recibir el U_SA local, con (5), la lista de los hosts que confirmaron el permiso, sólo elige al primero de la lista creando para éste un *p_espera* que se encargará de realizar la RPC al agente elegido.

pide_rta

En la figura I.10 se ilustra el esquema seguido por la operación *pide_rta*.

El U_SA local hace el llamdo a la RPC, originando (1), con el host donde se encuentre éste, y con la dirección del procedimiento que es obtenida de la lista de solicitudes. Indica también un tiempo muy pequeño de espera para evitar quedarse bloqueado, y los parámetros de salida que se esperan como resultado.



PDU's de "pide_rta"

(1) LLamada al RPC: nombre del host, direccion del procedimiento, timeout

Fig. I.11 Intercambio de mensajes para la operación pide_rta

En consecuencia el *p_espera* contesta con (2) enviando el resultado en los parámetos de salida.

En el caso de que una solicitud involucre muchas respuestas (por haberla solicitado a un grupo) se pide la respuesta al primer *p_espera* registrado en la lista que está contenida en la lista *listas_solic*. En caso de que conteste con la respuesta se elimina de dicha lista y el *p_espera* se autodestruye.

dest_solic

En la figura I.11 se ilustra el esquema seguido por la operación *dest_solic*.

En el caso de ésta operción, el U_SA local obtiene la lista de los *p_espera*, de *listas_solic*, que están asociados a la solicitud que se desea eliminar.

Posterirormente, hace una RPC a cada uno de los *p_espera*, esto es, envía a (1). Dicha RPC va dirigida a un procedimiento que realiza la operación de *exit*, y de ésta manera se consigue destruir todos los procesos de espera para una determinada solicitud.

VI. Manera de uso de las operaciones del servicio de Asociación

En ésta sección se describe la manera de uso de las operaciones implementadas, para ésto se proporciona una descripción de la operación, su sintaxis, el tipo de sus parámetros y el resultado que regresan. Todas las operaciones sobre grupos es unicamente para aquellos que son sin

jerarquía.

crea_gpo

Descripción:
Crea un grupo. El agente que la invoca se convierte en el dueò del grupo creado.

Sintaxis:
*char *crea_gpo()*

Regresa:

- *identificador del grupo*
- *-1.-* Error, no pudo crear grupo.

agrega

Descripción:
Agrega un elemento al grupo indicado. Ésta operación sólo puede ser realizada por el dueño del grupo.

Sintaxis:
*int agrega(char *grupo,char *elemento)*

Parámetros:

- grupo.- Es id del grupo.
- elemento.- Es el id del elemento que se pretende agregar.

Regresa:

- *0.-* Éxito.
- *-1.-* Error, no tiene datos del grupo.

- *-2.-* Error, no tiene datos del agente, es decir , del elemento
- *-3.-* Error, operación permitida sólo al dueño.
- *-4.-* Error, el elemento está inactivo.
- *-5.-* Error, el grupo está inactivo.

elimina

Descripción:
Elimina un elemento de un grupo. Ésta operación sólo puede ser realizada por el dueño del grupo.

Sintaxis:
*int elimina(char *grupo,char *elemento)*

Parámetros:

- grupo.- Es el id del grupo.
- elemento.- Es el id del elemento que se desea eliminar.

Regresa:

- *0.-* Éxito.
- *-1.-* Error, no tiene datos del grupo.
- *-2.-* Error, no tiene datos del agente, es decir , del elemento
- *-3.-* Error, operación permitida sólo al dueño.
- *-4.-* Error, el elemento está inactivo.
- *-5.-* Error, el grupo está inactivo.

esta_elem

Descripción:
Determina si un elemento se encuentra en un grupo.

Sintaxis:
*int esta_elem(char *grupo,char *elemento)*

Parámetros:

- grupo.- Identificador del grupo.
- elemento.- Identificador del elemento.

Regresa:

- *1.-* Si está.
- *0.-* No está.
- *-1.-* Error, no tiene datos del grupo.
- *-2.-* Error, no tiene datos del agente.

num_elem

Descripción:
Determina el número de elementos de un grupo.

Sintaxis:
*int num_elem(char *grupo)*

Parámetros:

- grupo.- Es el id del grupo.

Regresa:

- *>=0.-* Número de elementos.
- *-1.-* Error, no tiene datos del grupo.
- *-2.-* Error, grupo inactivo.

gpos_parti

Descripción:
Informa de los grupos en los que participapa el agente que invoca la operación.

Sintaxis:
*char *gpos_participo()*

Regresa:

- *lista de los grupos en los que participa*
- *0.-* No participa en ningún grupo.

gpos_dueño

Descripción:
Proporciona los grupos de los que es dueño el agente que invoca la operación.

Sintaxis:
*char *gpos_duenio()*

Regresa:

- *lista de los grupos que es dueño*
- *0.-* No es dueño de ningún grupo.

habilita

Descripción:
El agente que la invoca permite que se realice un determinado número de accesos remotos, a una función determinada, por un grupo o un agente remoto.

Sintaxis:
*int habilita(char *funcion,char *agente,int numero)*

Parámetros:

- funcion.- Es el id de la función.
- agente.- Es el id del agente o del grupo.
- numero.- Es el número de accesos.

Regresa:

- *0.-* Éxito.
- *-1.-* Error, función desconocida.

deshabilita

Descripción:
El agente que la invoca designa una desminución en el número de permisos para el accesos de una de sus funciones para un grupo o un agente.

Sintaxis:
*int deshabilita(char *funcion,char *agente,int numero)*

Parámetros:

- funcion.- Es el id de la función.
- agente.- Es el id del agente o del grupo.

- numero.- Es el número de accesos.

Regresa:

- *0.-* Éxito.
- *-1.-* Error, función desconocida.

dame_info

Descripción:
Proporciona, al agente que lo invoca, el primer mensaje recibido que no ha sido leído.

Sintaxis:
*char *dame_info()*

Regresa:

- *un mensaje de información.-*
- *0.-* No hay mensajes.

informa_gpo

Descripción:
Manda un mensaje de información a un grupo.

Sintaxis:
*int informa(char *grupo,char *informacion)*

Parámetros:

- grupo.- id del grupo.
- infromacion.- mensaje de información.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, no tiene datos del grupo.
- *-2.*- Error, grupo inactivo.

informa_uno

Descripción:
Manda un mensaje de información a un agente.

Sintaxis:
*int informa_agente(char *agente,char *informacion)*

Parámetros:

- agente.- Id del agente.
- informacion.- Mensaje de información.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, no tiene datos del agente.

pon_func

Descripción:
El agente que la invoca registra, de manera particular, el identificador y el número de procedimiento de una nueva función.

Sintaxis:
*int pon_funcion(char *funcion,int num_proc)*

Parámetros:

- funcion.- Identificador de la función.
- num_proc.- Número de procedimiento que tiene asociado.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, función repetida.
- *-2.*- Error, número de procedimiento repetido.

quita_func

Descripción:
El agente que la invoca elimina de su registro interno una función.

Sintaxis:
*int quita_func(char *funcion)*

Parámetros:

- funcion.- Identificador de la función.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, función inexistente.

solicita

Descripción:
Solicita a un grupo o agente remoto un llamado a una de sus funciones. En el caso del grupo, todos los integrantes deberán atender dicho llamado. Ésta operación no es bloqueante.
Los filtros para los parámetros de entrada y de salida deberán ser los mismos que estan definidos para las RPCs de Sun.

Sintaxis:
*int solicita(char *servidor,char *funcion,int timeout, xdrproc_t xdr_ent,void *p_ent, xdrprac_t xdr_sal,void *p_sal)*

Parámetros:

- servidor.- Identificador del grupo o del agente que atenderá la solicitud.
- funcion.- Identificador de la función solicitada.
- timeout.- Máximo tiempo que esperará la o las respuestas.
- xdr_ent.- Filtro xdr para los parámetros de entrada.
- p_ent.- Apuntador a la estructura que contiene los parámetros de entrada.
- xdr_sal.- Filtro xdr para los parámetros de salida.
- p_sal.- Apuntador a la estructura que contiene los parámetros de salida.

Regresa:

- *>=0.-* Es el número de solicitud.
- *-1.-* Error, no tiene datos del agente o grupo.
- *-2.-* Error, no atenderán la solicitud por no tener permiso el solicitante o por estar inactivado el grupo, y/o los agentes.

solicita_uno

Descripción:
Solicita a un grupo un llamado a un procedimiento remoto. Sólo uno de los integrantes lo atenderá . Esta operación no es bloqueante.
Los filtros para los parámetros de entrada y de salida deberán ser los mismos que estan definidos para las RPCs de Sun.

Sintaxis:
*int solicita_uno(char *grupo,char *funcion,int timeout, xdrproc_t xdr_ent,void *p_ent, xdrproc_t xdr_sal,void *p_sal)*

Parámetros:

- grupo.- Identificador del grupo que atenderá la solicitud.
- funcion.- Identificador de la función solicitada.
- timeout.- Máximo tiempo que se esperará la o las respuestas.
- xdr_ent.- Filtro xdr para los parámetros de entrada.
- p_ent.- Apuntador a la estructura que contiene los parámetros de entrada.
- xdr_sal.- Filtro xdr para los parámetros de salida.
- p_sal.- Apuntador a la estructura que contiene los parámetros de salida.

Regresa:

- *>=0.-* Es el número de solicitud.
- *-1.-* Error, no tiene datos del agente o grupo.
- -2.- Error, no atenderán la solicitud por no tener permiso el solicitante o por estar inactivado el grupo, y/o los agentes.

pide_rta

Descripción:
Pide una respuesta a una solicitud realizada previamente.
Los filtros para los parámetros de entrada y de salida deberán ser los mismos que estan definidos para las RPCs de Sun.

Sintaxis:
*int pide_rta(int solic,xdrproc_t xdr_sal,void *p_sal)*

Parámetros:

- solic.- Es el número de solicitud.
- xdr_sal.- Filtro para los parámetros de salida.
- p_sal.- Apuntador a la estructura que contiene los parámetros de salida.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, no existe el número de solicitud indicado.
- *-2.*- Error, no hay más respuestas.
- *-3.*- Error, no han llegado respuestas

dest_solic

Descripción:
Desecha todas las respuestas que han llegado y que esten por llegar para una solicitud dada.

Sintaxis:
int destruye_solic(int solic)

Parámetros:

- solic.- Es el número de solicitud.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, el número de solicitud no existe.

pon_agente

Descripción:
Agrega un agente a un registro interno que simula ser un directorio de agentes.

Sintaxis:
*int pon_agente(char *agente,char *dir_ip)*

Parámetros:

- agente.- Identificador del agente.
- dir_ip.- Dirección IP donde se encuentra el agente que se desea agregar.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, el identificador de agente ya existe.

quita_agente

Descripción:
Elimina un agente de un registro interno que simula ser un directorio de agentes.

Sintaxis:
*int quita_agente(char *agente)*

Parámetros:

- agente.- Es el identificador del agente.

Regresa:

- *0.*- Éxito.
- *-1.*- Error, agente inexistente.

VII. Instalación de la demo y del sistema

Para la instalación de la demo, y del sistema que comprende la IC, y el SAP; se requiere de la compilación de determinados archivos con C++. El sistema ha podido ser compilado en una Sun, Sparc Station 10, con Unix Berkeley versión 4.1.3. El compilador utilizado fue el g++ de GNU versión 2.5.8.
Para la compilación de los archivos se necesita ejecutar el siguiente shell:
$ compila
Éste generará cuatro archivos ejecutables:

1. demo
2. asoc
3. grupo
4. u_sa_demo

El primero es el que ofrece una demo utilizando una interface gráfica al usuario del *servicio de asociación*.
El segundo es el que proporciona el servicio de asociación. Mediante el tercero se crean grupos de trabajo. Y el último representa a el usuario del *servicio de asociación* (SA) y del *servidor de acciones privadas* (SAP).

La Demo

Para poner en operación la demo se requiere de los siguientes pasos:
Nodo local:

- Se requiere tener en un mismo directorio los siguientes archivos:
  - demo
  - asoc
  - grupo
  - ayuda.dat (contiene la información de la ayuda para la demo).
- Ejecutar los siguientes comandos:
  $ demo &
  $ asoc 7000
- Registrar los agentes remotos con los que se va a trabajar. Ésto se realiza desde la demo en el menú para poner agentes. La jerarquía del menú es:
  1. IC
  2. Servicio de Asociación
  3. Pon Agente

  De ésta manera se introducirá el identificador del agente y la dirección IP del host donde recida éste.
- Asegurarse que se tenga en el "/etc/hosts" el nombre del host con alias, para cada host donde recida un agente. Por ejemplo:
  132.248.204.1 uxmcc2 uxmcc2.iimas.unam.mx

Nodos remotos:

- Se requiere tener en un mismo directorio los siguientes archivos:
  1. u_sa_demo
  2. asoc
  3. grupo
  4. funciones.dat (contiene información de las funciones que ofrece el usuario local y/o remoto)
- ejecutar los siguientes comandos:
  $ u_sa &
  $ asoc 7000

El sistema

Para la instalación del sistema se requiere realizar el mismo procedimiento que para la demo con las diferencias siguientes:

1. El nodo local deberá tener y ejecutar un archivo diferente al de "demo". Éste archivo lo llamaremos "u_sa", y es aquel donde se utilizan las operaciones implemetadas. Más adelante se indicará la manera en que debe de ser construido.
2. En los nodos remotos también se deberá sustituir el archivo "u_sa_demo" por "u_sa".
3. Ni en el nodo local ni en los remotos se incluirá el archivo "funciones.dat".
4. No será necesario que esté el archivo "ayuda.dat" en el nodo local.
5. Registrar las funciones que ofrece el usuario local y/o remoto. Ésto se explicará en la siguiente sección.

Construcción del archivo "main_u_sa.c"

Mediante el archivo fuente "main_u_sa.c" se generará el ejecutable "u_sa",

realizando:
$ g++ -o u_sa main_u_sa.c
El archivo "main_u_sa.c" requiere:

1. Que se incluyan los archivos:
   - lib_ic.h
   - lib_sap.h
   - el archivo que tiene la definición de los parámetros de entrada y de salida de las funciones que ofrezca o solicite remotamente.
   - el archivo que contenga los filtros xdr para los parámetros de entrada y de salida.
   - el archivo "despacha_rpc.h" que será explicado más adelante.
   - el archivo que contenga las funciones que podrán ser accesadas remotamente.
2. Incluir la línea:
   atiende(DIRPROG,DIRVERS);
   dentro del cuerpo de "main". Ésta instrucción pone listo al programa para recibir cualquier llamada remota de uno de sus procedimientos.

Un ejemplo del archivo "main_u_sa.c" es:

```
// Nomarch: main_u_sa.c
// Descripcion: Prueba el usuo de las librerías del Usuario del servicio
// de asociacion y del servidor de acciones privadas.

#include "lib_ic.h" // librerías para el uso de la IC
#include "lib_sap.h" // librerías para el uso del SAP

#include "parametros.h" // Definición de los parámetros de entrada
// y de salida de las funciones que ofrezco
#include "filtros-rpc.h" // Filtros para los parámetros de entrada y salida
#include "despacha-rpc.h" // Case para que el servidor atienda la
// función requerida.
```

```
// También tiene las definiciones de los num de proced.
#include "func-rpc.h" // Son las funciones que ofrezco.

main()
{
atiende(DIRPROG,DIRVERS);
return 0;
}
```

Elaboración del archivo "despacha-rpc.h"

Éste archivo tiene como propósito determinar que función se está solicitando y que parámetros de salida, de entrada y qué filtros se usarán para llevar a cabo la ejecución de ésta.

Como ejemplo del archivo "despacha-rpc.h" tenemos el siguiente:

```
// Nombre: despacha-rpc.h

#ifndef DESPACHA_RPC_H
#define DESPACHA_RPC_H

   // Definiciones de los numeros de procedimiento:
#define SUMA 1 /* num. del procedimiento*/
#define RESTA 2 /* num. del procedimiento*/

int despacha(struct svc_req *rqstp,SVCXPRT *transp)
{
void *p_sal; // no modificar
extern void obten_p(SVCXPRT *transp,xdrproc_t xdr_ent,void *p_ent);
// no modificar
extern void manda_p(SVCXPRT *transp,xdrproc_t xdr_sal,void *p_sal);
// no modificar
void *dame_rta(int *p_ent); // no modificar
extern int FUNC; // no modificar

extern int *suma(struct parament_suma *p_ent_suma);
extern float *resta(struct parament_resta *p_ent_resta);
```

```
struct parament_suma p_ent_suma;
struct parament_resta p_ent_resta;

switch (rqstp- >rq_proc)
{

case NULLPROC: exit(0); // no modificar
               break;
case DAME_RTA: p_sal= dame_rta(0);
               if(FUNC==SUMA)
               manda_p(transp, xdr_int, p_sal);
               if(FUNC==RESTA)
               manda_p(transp, xdr_float, p_sal);
               break;
case SUMA:     obten_p(transp, filtro_sal_suma, &p_ent_suma);
               p_sal_suma= suma(&p_ent_suma);
               manda_p(transp, filtro_sal_suma, p_sal);
               break;
case RESTA:    obten_p(transp, filtro_sal_resta, &p_ent_resta);
               p_sal_resta= resta(&p_ent_resta);
               manda_p(transp, filtro_sal_resta, p_sal);
               break;
default:       svcerr_noproc(transp);
               return;


}
}
#endif
```

En el ejemplo tenemos definidas las funciones: suma() y resta() las cuales tienen el número de procedimiento 1 y 2 respectivamente.
La suma toma dos valores enteros como entrada y regresa un entero; la resta, en cambio, toma dos reales. De esta manera se debe tener en el archivo de "parametros.h" las siguientes definiciones:

```
typedef struct param_ent_suma{
int x; /* primer operando */
int y; /* segundo operando */
} entrada_suma;

typedef struct param_ent_float{
float x; /* primer operando */
float y; /* segundo operando */
} entrada_resta;
```

Todos los parámetros de salida serán del tipo:
void *p_sal;
En el *switch* se tienen las posibles funciones requeridas. La primera cuya constante es NULLPROC indica que se ha requerido que el programa ejecute *exit*. La segunda, cuya constante es DAME_RTA, solicita que se proporcione la respuesta a una RPC que ésta mando hacer y cuyo resultado lo almacenó en: p_sal.
Las demas constantes corresponden a los procedimientos que ofrece el servidor, en éste caso son el de suma y resta.

La variable externa FUNC identifica el número de procedimiento del que se está requiriendo la respuesta. Y ésto sucede cuando el U_SA le pide la respuesta a un *p_espera*.

La función obten_p() permite obtener los parámetros de entrada provenientes del cliente. Y la función manda_p() permite mandar los parámetros de salida al cliente.

Registro de las funciones

Las funciones que ofrece el U_SA remoto y/o local, deberán ser registradas, en lo que simula ser un directorio particular de funciones. Lo anterior se realiza con la operación:
*int pon_funcion(char *func,int num_func)*
donde *func* es el identificador de la función y *num_func* es el número de función dentro del programa.
En el ejemplo anterior la operación de "suma" tiene el número de

función igual a 1. Y su identificador de función puede ser por ejemplo "Suma".

Bibliografía

[BG88] Alan H. Bond and Les Gasser, editors. *Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufmann Publishers, INC, 1988.

[BSS91] Henri E. Bal, Jennifer G. Steiner, and Andrews S. Programming languages for distributed computing system. *ACM Computing Surveys*, 23(1):261–322, 1991.

[CZ85] David R. Cheriton and Willy Zwaenepoel. Distributed process groups in the v kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, 1985.

[D.85] Gelernter D. Generative communication in linda. *ACM Trans. Prog. Lang. Syst.* 7(1):80–112, 1985.

[DS88] Rendal Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 333–356, 1988.

[GC85] Agha Gul and Hewitt Carl. Concurrent programming using actors. In Akinori Yonezawa and Mario Tokoro, editors, *Object-Oriented Concurrent Programming*, pages 37–53, 1985.

[Hen85] Lieberman Henry. Concurrent object-oriented programming act 1. In Akinori Yonezawa and Mario Tokoro, editors, *Object-Oriented Concurrent Programming*, pages 9-35, 1985.

[HI91] Carl Hewitt and Jeff Inman. Dai betwixt and between: From inteligent agents to open systems science. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, 21(6):1409–1419, 1991.

[Hoa78] Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[HRD85] Durfee Edmund H., Lesser Victor R., and Corkill Daniel D. Coherent cooperation among communicating problem solvers. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 268–284, 1985.

[LB92] Gasser Les and Jean-Pierre Briot. Object-based concurrent programming. In N.M. Avouris and Les Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–107, 1992.

[LCN88] Gasser Les, Biaganza Carl, and Herman Nava. Implementing distributed ai systems using mace. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 445–450, 1988.

[LE80] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and experiment. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 120–139, 1980.

[R.91] Andrews Gregory R. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, 1991.

[RB83] Andrews Gregory R. and Schneider Fred B. Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15(1):3–43, 1983.

[S.80] Fox Mark S. An organizational view of distributed systems. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 140–150, 1980.

al] Ma. Margarita Catalan Salgado. *Servicios de directorio y de asociacion de la interfaz de comunicacion para la cooperacion entre sistemas abiertos.* Tesis Lic. en Informatica. Inst. Tec. del ISMO SEP y Laboratorio Nacional de Informatica Avanzada, AC. 1994.

ho90] Shoham. *Agent-Oriented Programming.* STAN-CS-1335-90, Robotics Laboratory, Computer Science Department, Stanford University, 1990.

ho86] Mallone Thomas. Modelling coordination in organizations and markets. In Alan H. Bond and Les Gasser, editors, *Distributed Artificial Intelligence*, pages 151–158, 1986.

/SJ88] Zachary W., Robertson S., and Black J.(Eds). *Introduction to Cognition, Computation, and Cooperation.* Albex Publishing Corp., Norward, NJ, 1988.